НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)
Автоматизованих систем обробки інформації і управління
(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Проблемно-орієнтовна мова для програмування задач з біоінформатики*

Виконав: студент IV курсу, групи *ІП-61Каджая Володимир Миколайович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *доц., к.т.н., Баклан І.В.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

доц., к.т.н., Кисленко Ю.І.
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Каджая Володимиру Миколайовичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Проблемно-орієнтовна мова для програмування задач з біоінформатики»*

керівник проєкту *Баклан Ігор Всеволодович, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «08» *червня 2020 року*

3. Вихідні дані до проєкту

Технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог

2) Опис розробленої мови програмування: типи та структури даних, система типізації, семантика

3) Конструювання програмного забезпечення: синтаксичний та лексичний аналізатори, семантичний аналізатор, AST-дерево

4) Аналіз якості та тестування програмного забезпечення: аналіз якості, опис

процесів тестування, опис контрольних прикладів

5) Впровадження та супровід програмного забезпечення: робота з мовою програмування, супровід та розгортання програмного забезпечення

5. Перелік графічного матеріалу

1) Схема бізнес-процесу

2) Схема структурна станів системи

3) Схема структурна класів програмного забезпечення

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*17.03.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*24.03.2020*	
3.	*Постановка та формалізація задачі*	*27.03.2020*	
4.	*Аналіз вимог до програмного забезпечення*	*03.04.2020*	
5.	*Алгоритмізація задачі*	*10.04.2020*	
6.	*Моделювання програмного забезпечення*	*17.04.2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*24.04.2020*	
8.	*Розробка архітектури програмного забезпечення*	*30.04.2020*	
9.	*Розробка програмного забезпечення*	*08.05.2020*	
10.	*Налагодження програми*	*15.05.2020*	
11.	*Виконання графічних документів*	*20.05.2020*	
12.	*Оформлення пояснювальної записки*	*27.05.2020*	
13.	*Подання ДП на попередній захист*	*28.05.2020*	
14.	*Подання ДП рецензенту*	*09.06.2020*	
15.	*Подання ДП на основний захист*	*17.06.2020*	

Студент _____ Володимир КАДЖАЯ
(підпис)

Керівник _____ Ігор БАКЛАН
(підпис)

№з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-6111.045490.01.81	Пояснювальна записка	78	
3	А4	КПІ.ІП-6111.045490.02.91	Технічне завдання	12	
4	А4	КПІ.ІП-6111.045490.03.51	Програма та методика тестування	6	
5	А3	КПІ.ІП-6111.045490.04.33	Керівництво програміста	5	
6	А3	КПІ.ІП-6111.045490.05.13	Опис програми	14	
7	А3	КПІ.ІП-6111.045490.06.99 СС	Схема структурна стану	1	
8	А3	КПІ.ІП-6111.045490.06.99	Схема структруна бізнес-процесу	1	
9	А3	КПІ.ІП-6111.045490.06.99 СС	Схема структур класів програмного забезпечення	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6111.045490.00.90		
Розробн.		Каджая В.М.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Баклан І.В.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-61	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

До пояснювальної записки входять 5 розділів, 9 таблиць, 13 рисунків, а також 21 посилань на джерела інформації та додаток з програмним кодом. Загальний обсяг складає 120 сторінок.

Метою дослідження є проектування мови програмування для моделювання експериментів із злиття клітин.

У першому розділі наведено основні теоретичні відомості щодо граматики мови та опис її реалізації у мові Kujira, а також виведено основні функціональні вимоги до мови програмування.

У другому та третьому розділі наведено опис мови програмування, способів її роботи, а також архітектурні та алгоритмічні деталі роботи комплексу.

У четвертому розділі наведені тестування результату роботи та описані способи тестування комплексу у різних умовах та середовищах, а у п’ятому розділі описано спосіб його розгортання.

Також до пояснювальної записки додаються графічні матеріали із схемою діаграми діяльності, схемою структурною класів програмного забезпечення та схемою використання.

КЛЮЧОВІ СЛОВА: lark, LLVM, LL, AST-ДЕРЕВО, ІНТЕРПРЕТАТОР, КОМПІЛЯТОР, МОВА ПРОГРАМУВАННЯ, ЛЕКСИЧНИЙ АНАЛІЗАТОР, ПАРСИНГ

ABSTRACT

The explanatory note includes 5 sections, 9 tables, 13 figures, as well as 21 links to sources of information and an application with program code. The total volume is 120 pages.

The aim of the study is to design a programming language for modeling cell fusion experiments.

The first section provides basic theoretical information on the grammar of the language and a description of its implementation in the Kujira language, as well as the main functional requirements for the programming language.

The second and third sections describe the programming language, methods of its operation, as well as architectural and algorithmic details of the complex.

The fourth section describes the testing of the result of the work and describes the methods of testing the complex in different conditions and environments, and the fifth section describes the method of its deployment.

Graphic materials with a diagram of the activity diagram, a diagram of the structural classes of the software and a diagram of the use are also attached to the explanatory note.

KEYWORDS: lark, LLVM, LL, AST-TREE, INTERPRETER, COMPILER, PROGRAMMING LANGUAGE, LEXICAL ANALYSIS, PARSING

Пояснювальна записка
до дипломного проєкту

на тему: Проблемно-орієнтовна мова для програмування задач з біоінформатики

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AST (Abstract Syntax Tree) – позначене орієнтоване дерево, в якому вершини співставлені з операторами мови програмування, а листя – з операндами, універсальний спосіб проміжного представлення інструкцій мови програмування.

lark - бібліотека для Python.

LALR - являє собою розширення алгоритму SLR (1). У ряді випадків працює тоді, коли побудова SLR (1) таблиці розбору для даної граматики неможливо через конфлікти зрушення-згортка або згортка-згортка. Таким чином, клас граматик, що розбираються по LALR (1) ширше, ніж клас SLR (1) - граматики.

LR - синтаксичний аналізатор для вихідних кодів програм, написаних на деякій мові програмування, який читає вхідний потік зліва (Left) направо і виробляє найбільш праву (Right) продукцію контекстно-вільної граматики. Використовується також термін LR (k) -аналізатор, де k висловлює кількість непрочитаних символів предпросмотра у вхідному потоці, на підставі яких приймаються рішення при аналізі. Зазвичай k дорівнює 1 і часто опускається.

LLVM - проект програмної інфраструктури для створення компіляторів і супутніх їм утиліт. Складається з набору компіляторів з мов високого рівня (так званих «фронтенда»), системи оптимізації, інтерпретації і компіляції в машинний код. В основі інфраструктури використовується RISC-подібна платформо незалежна система кодування машинних інструкцій (байткод LLVM IR), яка являє собою високорівнева асемблер, з яким працюють різні перетворення.

JIT - технологія збільшення продуктивності програмних систем, що використовують байт-код, шляхом компіляції байт-коду в машинний код або в інший формат безпосередньо під час роботи програми. Таким чином досягається висока швидкість виконання в порівнянні з інтерпретується байт-

кодом (порівняння з компільовані мови) за рахунок збільшення споживання пам'яті (для зберігання результатів компіляції) і витрат часу на компіляцію.

STL - набір узгоджених узагальнених алгоритмів, контейнерів, засобів доступу до їх вмісту і різних допоміжних функцій в C ++.

LL - в інформатиці спадний синтаксичний аналізатор для деякого підмножини контекстно-вільних граматик, відомих як LL-граматики. При цьому не всі контекстно-вільні граматики є LL-граматиками. Букви L в вираженні «LL-аналізатор» означають, що вхідний рядок аналізується зліва направо (left to right), і при цьому будується її лівобічний висновок (leftmost derivation).

ВСТУП

Перші програми полягали в установці перемикачів на панелі ЕОМ. Таким чином можна було писати невеликі програми. Згодом програмісти почали писати програми за допомогою машинного коду. Таким чином можна було писати більш складні програми, всі ці програми були якимись інструкціями для комп'ютера. Але тут теж є свої недоліки так як потрібно було контролювати пам'ять комп'ютера, а так само складність написання програм, особливо введення-виведення. Тому з'явився перший машинно-орієнтована мова - Асемблер. Програмісти могли тепер прописувати команди які переводилися в машинний код і виконувалися комп'ютером. Через деякий час з'явиться перший високорівнева мова програмування - FORTRAN. Ці мови будуть зрозумілі людині, вони також як і Асемблер буде транслювати всі в машинний код і запускатися комп'ютером.

Відповідно до стандарту ANSI / IEEE 1471 2000 року архітектура програмного забезпечення: «Фундаментальна організація системи, втілена в її компонентах, їх відносинах один з одним і з навколишнім середовищем, а також принципи, що визначають її дизайн і розвиток».[10, 13]

Тому компоненти моделюються «протягом усього життєвого циклу розробки та послідовно переглядаються на розгортання та час виконання». [11, 13].

У нашому сучасному світі щороку з'являється нова мова програмування, який займає певну нішу в вузькій області. Наприклад цього може послужити сучасну мову програмування використовується зараз компанією Google для мобільного розробці - Kotlin або від компанії Apple нову мову програмування Swift.

Вже близько 60 років компілятори і інтерпретатори є актуальними в наші дні. За допомогою них ми можемо спілкуватися з комп'ютером. Щороку виникають все нові і нові компілятори, поліпшується процес обробки, підвищується продуктивність. Наприклад зараз популярними стали динамічні

компілятори типу JIT (just in time), а також більш поліпшені віртуальні машини такі як LLVM (Low-Level Virtual Machine). [19]

Оригінальною мовою обчислювальних технологій було Fortran, скорочене до «Formula Translating System», випущеної у 1957 році. З тих ранніх часів вчені мріяли написати висококласні загальні формули та автоматично перекладати їх на низькорівневий ефективний код, з урахуванням конкретних типів даних, до яких потрібно застосувати формули. Напередодні історичні кроки до досягнення цієї мрії, і її домінування в багатьох областях є свідченням її успіху[1].

Мови змінювались роками. Сучасні наукові обчислювальні середовища, такі як Python, MATLAB, R, Julia, Octave, Scilab[2]. Але усі ці мови програмування були розроблені окремо для конкретної мети. Тому було вигадано розробити мову програмування яка б об'єднала б усі функціональні переваги.

Kujira - мова яка використовується у біоінформатиці але є стандартні бібліотеки для обчислення математичних формул.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Змістовний опис і аналіз предметної області

Мова програмування - це штучна мова створена для того щоб взаємодіяти, "спілкуватися" з комп'ютером.

Мови діляться на 2 типи: високорівневі і низькорівневі.

Високорівневі мови - це мови які зрозумілі для сприйняття людині, в свою чергу низькорівневі мови програмування близькі до машинного коду. Також мови мають різні парадигми такі як: функціональна, об'єктно-орієнтована, процедурна або аспектно-орієнтована і т.д.

Мова програмування Kujira ставитися на даний момент до функціонального мови і є високорівневим. Дана мова була створена для того щоб об'єднати кращі принципи різних мов програмування. На даний момент він має лямбда вираз.

Таблиця 1.1 – Порівняння основних функціоналів у Kujira та R

Kujira	R programming language
робота з файлами (читання, написання)	робота з файлами (читання, написання)
робота з ДНК та клітинами	робота з ДНК та клітинами
робота з великими числами, а також деякі арифметичні операції	робота з великими числами, а також деякі арифметичні операції
logging	logging
сортування	сортування
unit tests	unit tests
використання потоку	використання потоку

Мова програмування R як і Kujira мають багато спільного і у таблиці 1.1 наведені деякі приклади. Але R має деякі недоліки. По перше відносна складність у використанні для користувача, незнайомого з мовами програмування. По друге R частіше використовують в bigdata та штучному інтелекту так як має багато бібліотек для аналізу даних але не має гарних бібіліотек з біоінформатики. По третє R не швидка мова програмування і іноді треба об'єднувати або переписувати деякі програми на більш низькорівневих мовах таких як C++.

Kujira на сам перед намагається об'єднати усі мови такі як R, Julia, Python у єдине ціле.

Приклад конвертації ДНК у протеїн на мові Kujira та R:

```
translate_to_protein("file.txt")
dna1 <- DNAString("TTGATATGGCCCTTATAA")
translate(dna1)
```

1.2 Огляд наявних аналогів

На сьогодні імплементації програмування задач з біоінформатики існує у вигляді бібліотек.

1.2.1 Perl

Bioperl, мабуть, найуспішніший проект, який є частиною O | B | F. Bioperlcode - це велика бібліотека основних модулів, написаних на Perl для підтримки обробки, обробки і управління біологічної інформацією у вигляді послідовностей [14]. Одна з причин, по якій була вибрана мова програмування Perl, полягала в тому, що він уже набув популярності в співтоваристві біоінформатики за підтримку процесу обробки тексту і зіставлення зі зразком. Проект Bioperl намагається емулювати об'єктно-орієнтовану парадигму програмування, використовуючи модулі Perl і дотримуючись трьох принципів проектування. Перший принцип - відокремити інтерфейс від реалізації. Другий

принцип полягає в наданні базової структури для відповідної операції шляхом узагальнення загальних підпрограм в один модуль. Третій і останній принцип - використовувати шаблони фабрики і стратегії, визначені Еріхом Гаммом.

Недоліком Bioperl є те що існує багато способів використання BioPerl, від простих сценаріїв до дуже складних об'єктних програм. Це робить мову незрозумілим і іноді важким для розуміння. Для стількох модулів, що є в BioPerl, деякі не завжди працюють так, як вони призначені.

1.2.2 Python

В цілому, скриптові мови високого рівня є популярним вибором для дослідників в галузі біоінформатики. В доповнені до своїх можливостей в якості мови сценаріїв, Python має додаткову підтримку розширених числових можливостей в рамках проекту Scientific Tools for Python (SciPy) [6,14]. Проект Biopython був створений в 1999 році і змодельований за зразком успішного проекту Bioperl [15]. Велика частина роботи проекту Biopython була зосереджена на створенні парсерів для біологічних даних і розробці корисного інтерфейсу для подання послідовностей. Однією з унікальних особливостей проекту Biopython є використання нестандартного подієво-орієнтованого дизайна парсера.

1.2.3 Julia

Проект BioJulia - це проект з відкритим вихідним кодом, спрямований на створення інфраструктури для біоінформатики на мові програмування Julia. Він спрямований на надання швидких і доступних програмних бібліотек. Центральним пакетом, розробленим в рамках проекту, є Bio.jl, який надає основні функції, включаючи біологічні символи / послідовності, аналізатори формату файлів, алгоритми вирівнювання, обгортки для зовнішніх програм і тощо. Він також підтримує декілька поширених форматів файлів, таких як FASTA, FASTQ, BED, PDB.[12] Недоліком є те що з’явилась ця бібіліотека

відносно недавно, тому документація до бібліотеки не до кінця написана, також сама бібліотека ще розроблюється.

1.2.4 Java

BioJava є бібліотекою Java з відкритим вихідним кодом і є частиною O | B | F [14]. BioJavaproject в першу чергу стосується того, як представляти послідовності. Головна особливість проекту BioJava полягає в тому, що в мові програмування Java визначені дві унікальні схеми уявлення послідовностей. Перша схема є базовим уявленням рядки токена і використовується, коли анотація не важлива при аналізі даних. Друге подання є анотована структура послідовності і використовується, коли потрібно повністю анотоване уявлення послідовності.

1.2.5 Haskell

Haskell - це чисто функціональна мова програмування загального призначення. Haskell використовувався дослідницькою групою Роберта Гігеріха для реалізації алгоритмів динамічного програмування, в тому числі задіяних в граматиках згортання РНК [14]. Група стверджує, що їх робота додає значну гнучкість і універсальність при розробці нових алгоритмів динамічного програмування. Але недоліком є важкість розуміння синтаксису мови та деякі бібліотеки мають нестачу в підтримці із-за чого може виникнути помилки.

1.3 Аналіз вимог до програмного продукту

1.3.1 Розроблення функціональних вимог

Мова програмування Kujira орієнтовна перш за все на використання технічними спеціалістами, а функціональні вимоги до мови описуються з точки зору характеристик та властивостей результату роботи.

Схема варіантів використання наведено на рисунку 1.1.

Рисунок 1.1 – Схема варіантів використання

Більш детально у розгорнотому виді можна побачити на рисунку 1.2.

Рисунок 1.2 - Схема варіантів використання розгорнутий вид

Таблиця 1.3 Функціональні вимоги

Варіант використання	Функціональна вимога	Пріоритет
Написання програми	Мова повинна мати лексичні та синтаксичні правила	Високий
	Мова повинна мати чітке визначення семантики	Високий
	Мова має чітко описану систему типів та структур даних	Високий
	Мова не повинна залежати від середовища або операційної системи	Високий
	Мова надає базові модулі (такі бібліотеки як bioinf) для використання їх у програмних продуктах	Високий
Виконання програми на розробленій мові	Під час запуску програми перевіряється на синтаксичні та лексичні помилки. Якщо вони присутні відображається відповідна помилка	Високий

Продовження таблиці 1.3

	Коли модуль імпортується, Kujira виконує весь код у файлі модуля, якщо такий файл існує інакше видає помилку	Високий
	оператор import виконує пошук за списком шляхів в sys.path	Високий
	sys.path завжди містить шлях до скрипту, що викликається в командному рядку, і не залежить від робочого каталогу в командному рядку.	Високий

1.4 Цілі та задачі розробки

Ціль дипломної роботи - створення мови програмування та засобів її виконання для написання програм в області біоінформатики.

Для досягнення мети були вирішені наступні задачі:

- розробка дизайну мови (синтаксис мови та граматика);
- розробка структур даних та систему типів;
- лексичний та синтаксичний аналіз;
- відносно висока швидкість компіляції програм;

- використання у будь-якій операційній системі;

1.4.1 Розроблення нефункціональних вимог

Нефункціональні вимоги визначають системні критерії; вони не визначають функціональність, а скоріше основи системи, що допоможе і дасть змогу деталізувати архітектуру системи. Щоб мати можливість аналізувати і аналізувати проект, ці вимоги будуть розділені на категорії.

1.4.1.1 Доступність

– Мова програмування Kujira має бути доступна через веб-браузер завдяки якій мовжна встановити мову для настільного комп'ютера або ноутбуку.
– ОС, які будуть підтримувати мову прогамування Kujira це – Windows, Unix-системи та MacOS X.
– ОС, які будуть підтримувати мову програмування Kujira повний досвід, - це їхні останні версії Python 3.

1.4.1.2 Документація

– Вся документація буде розроблена в інструменті Confluence для онлайн-співпраці. Кожен розробник буде додавати документацію в міру розвитку проекту.
– Документація буде доступна зацікавленим сторонам з можливістю експорту в різних форматах, таких як PDF або Word Doc.

1.4.1.3 Розгортання

– Незважаючи на те, що технології, які використовуватиме мова програмування, є стерпним, система буде побудована в операційній системі на основі Unix, і деякі функції будуть підтримуватися тільки цією ОС.
– Для контролю версій програмного забезпечення SVN буде використовувати дотримуючись кращих практик управління версіями

управління. Це включає в себе виробництво, забезпечення якості, постановку або будь-який інший контроль або не середовище.

1.4.1.4 Платформа та мова програмування

– Система буде розроблятися в середовищі на основі Unix, середовища на основі системі Windows також будуть перевірені, але гарантію правильності працювання не даєм. Також вимогою є встановлення отсаньої версії Python та C++.

1.5 Висновки по розділу

В результаті аналізу предметної області було оглянуто основні пооження предметного середовища та проаналізовано можливі алгоритми у біоінформатиці, які будуть використовуватися для розробки мови програмування.

Також було проведене порівняння конкурентів, а саме актуальних реалізацій мов(бібліотек та фреймворків). Для кожного з наведених аналогів було перелічено їх переваги, які допомогли у досягненні результату розробки, а також недоліки, які були враховані та усунені під час реалізації.

На основі загальних відомостей на наявних рішень сформована основна мета розробки та цілі, які необхідно переслідувати для її досягнення. Крім того, описано задачі, що будуть розв’язані для отримання бажаного результату.

2 ОПИС РОЗРОБЛЕННОЇ МОВИ

2.1 Граматика

Перш ніж почати писати будь-яку мову програмування, потрібно проаналізувати те яким синтаксично буде виглядати майбутня мова програмування, і створити граматику даної мови. Як створюється граматика? Найпростішим і ефективним способом є використання розширеної форми Бекуса-Наура. Розширена форма Бекуса-Наура - це формальна система визначення синтаксису мови. Іншими словами, всі ті змінні цілочисельного типу (int), десяткового (double, float), масиви тощо описується за допомогою розширеної форми. Розширена форма Бекуса-Наура - це формальна система визначення синтаксису мови. Дана форма використовується для опису контекстно-вільних формальних граматик, зазвичай використовується для опису синтаксису мов програмування, форматів документів, наборів інструкцій і протоколів зв'язку. [1] Усі популярні мови програмування найчастіше використовують саме розширену нотацію. Також треба створити парсинг мови. Найчастіше зустрічаються такі алгоритми як LALR, LR, LL. Тепер розглянемо на прикладі мови Kujira. Для цього використовувався LALR. LALR або Lookahead LR parser - це спрощена версія канонічного LR парсеру [3]. Парсери LR - це ефективні аналізатори знизу вгору, які можуть бути створені для великого класу контекстно-вільних граматик. Граматика LR (k) - це граматика, яка генерує рядки кожен з яких може бути проаналізовано під час одного детермінованого сканування зліва направо, не заглядаючи вперед більше, ніж на k символів. Ці парсери, як правило, дуже ефективні і гарні в повідомленнях про помилки, але, на жаль, їх дуже важко написати без допомоги спеціальних програм [4]. У граматики LR є один недолік, у випадках, коли одна чи кілька станів. У таких прикладах синтаксичний аналізатор має конфлікт такий як "читання-зменшення", "зменшення-зменшення" або обидва. У першому випадку парсер не може вирішити, чи читати наступний символ вводу або

зменшувати фразу в стеці. В останньому випадку плутанина між різними скороченнями [5], для цього був придуманий LALR. За допомогою синтаксичного аналізу LALR (lookahead LR) ми намагаємося зменшити кількість станів в синтаксичному аналізаторі LR (1) шляхом об'єднання схожих станів. Це зменшує кількість станів до рівня, рівного SLR (1), але при цьому зберігає деяку потужність запитів LR (1) [6]. Ця система (LALR) дозволяє автоматично створити лексему мови.

Розглянемо на прикладі:

```
? start: calc | NAME "=" calc -> assign | value | printval
```

Це запис граматики в формі LALR. `?start:` це оператор який приймає числа, створює змінні, арифметичні операції і масиви, і вбудовані функції такі як print. Тепер розглянемо на прикладі моєї мови. Для цього я використовував lark. Ця система дозволяє автоматично створити лексему нашої мови.

Розглянемо на прикладі:

```
grammar = '''

?start: calc | NAME "=" calc -> assign | value | printval | func | pow_ | ceil_ | acos_ | asin_ | atan_ |
cos_ | exp_ | fabs_ | floor_ | sin_ | tan_ | sqrt_ | log_ | log10_ | importingfile | NEWLINE | condition

?calc: prod | calc "+" prod -> add | calc "-" prod -> sub

?prod: atom | prod "*" atom -> mul | prod "/" atom -> div

?atom: NUMBER -> number | "-" atom -> neg | NAME -> var | "("calc")"

?value: array | SIGNED_NUMBER -> number | string | sorting | sets

?printval: "print" value

?func: "func" "(" [NAME ":" value ("," NAME ":" value)*] ")" "{" printval "}"

?array: "[" [value ("," value)*] "]"

?sets: "{" [value ("," value)*] "}"
```

```
condition: "if" [statement "then" result ("elseif" statement ":" result)*] | "if" [statement "then" result
("elseif" statement ":" result)*] "else" result

if: "if"

then: "then"

elseif: "elseif"

else: "else"

statement: expression

result: printval | condition

expression: VARIABLE action_operator (VARIABLE | SIGNED_NUMBER)

VARIABLE: /[a-zA-Za-яA-Я0-9_.-]+/

?action_operator: ACTION_OPERATOR

ACTION_OPERATOR: "<"|">"|"="|"is"|">="|"<="|"!="|"in"|"equals"

?pow_ : "pow" "(" NUMBER "," NUMBER ")"

?ceil_ : "ceil" "(" NUMBER ")"

?acos_ : "acos" "(" NUMBER ")"

?asin_ : "asin" "(" NUMBER ")"

?atan_ : "atan" "(" NUMBER ")"

?cos_ : "cos" "(" NUMBER ")"

?exp_ : "exp" "(" NUMBER ")"

?fabs_ : "fabs" "(" NUMBER ")"

?floor_ : "floor" "(" NUMBER ")"

?sin_ : "sin" "(" NUMBER ")"

?tan_ : "tan" "(" NUMBER ")"

?sqrt_ : "sqrt" "(" NUMBER ")"

?log_ : "log" "(" NUMBER ")"

?log10_ : "log10" "(" NUMBER ")"
```

```
?sorting : "sort" "(" "[" [value ("," value)*] "]" ")"

importingfile: "import" NAME NAME  NUMBER "," NUMBER

string : ESCAPED_STRING

COMMENT: "//" /(.|\\n|\\r)*/

%import common.NUMBER

%import common.SIGNED_NUMBER

%import common.ESCAPED_STRING

%import common.CNAME -> NAME

%import common.WS_INLINE

%import common.NEWLINE

%ignore " "

%ignore WS_INLINE

%ignore COMMENT

'''
```

Більш детальніше можна дізнатись про команди та їх опис у таблиці команди граматики мови.

Таблиця 2.1 - Команди граматики мови

Назва команди	Опис
`start`	Базова команда. Реалізує єдиний метод
`calc`	Базова команда. Використовується для арифметичних операцій

Продовження таблиці 2.1

`atom`	Базова команда. Використовується для створення відмінних чисел
`value`	Базова команда. Використовується для виклику списку, створення змінної, та виклику деяких функцій
`printval`	Базова команда. Використовується для виведення інформації
`func`	Базова команда. Використовується для створення функцій
`array`	Базова команда. Створення списку
`sets`	Базова команда. Створення множин
`condition`	Базова команда. Використовується для побудови умовного оператору
`if`	Базова команда. Використовується для виклику ключового слова if
`then`	Базова команда.Використовується для виклику ключового слова then
`elseif`	Базова команда. Використовується для виклику ключового слова elseif
`else`	Базова команда. Використовується для виклику ключового слова else

Продовження таблиці 2.1

statement	Базова команда. Використовується для побудови виразу для умовного оператору
result	Базова команда. Використовується для виведення результату з умовного оператору
VARIABLE	Базова команда. Використовується для виклику змінних
action_operator	Базова команда. Використовується для арифметичних знаків таких як <, >, тощо
sorting	Базова команда. Використовується для сортування списку або інших структур даних
importingfile	Базова команда. Використовується для іпортування модулів
string	Базова команда. Використовується для створення строкового літералу
COMMENT	Базова команда. Використовується для відтворення коментарів
count_dna	Базова функція для підрахунку елементу у клітині ДНК
generate_dna_sequence	Базова функція для генерування ДНК

Продовження таблиці 2.1

dna_frequency_map	Базова функція для підрахунку частоти елементу у ДНК
read_dnafile	Базова функція для читання з файлу ДНК
translate_to_protein	Базова функція для перетворення ДНК у протеїн
mutate	Базова функція генерування мутації
reverse_complement	Базова функція для зворотнього доповнення
count_non_dna_bases_seq	Базова функція для обчислення баз не ДНК
dna_concat	Базова функція для злиття клітин
sequence_alignment	Базова функція для вирівнювання послідовностей – спосіб упорядкування послідовностей ДНК
motif	Базова функція для обчислення послідовності мотивів для аналізу генної регуляції
protein_mass	Базова функція для обчислення протеїну

2.2 Лексичний аналізатор

Лексичний аналізатор - це частина компілятора який читає вихідний файл і знаходить токени або лексеми. Прикладом може бути додавання двох чисел. Що він повинен зробити, так це визначити ідентифікатори, оператор присвоювання і числа, і оператор додавання. Так як використовується lark то мені не доводиться реалізовувати лексичний аналізатор з нуля. Якщо використовувати LLVM то для цього потрібно створювати аналізатор використовуючи *lex* або *Flex*.

2.3 Синтаксичний аналізатор

Синтаксичний аналіз - це дерево розбору коду або більш правильно зіставлення послідовності лексем з його формальної граматикою.

Розглянемо приклад.

Нехай вихідний код наш буде таким

```
print "hi"
```

Відповідно дерево нашого коду буде таким як на рисунку 2.1:

```
Tree(string, [Token(ESCAPED_STRING, '"hi"')])
```

Рисунок 2.1 - Синтаксичний аналіз функцшї

Як було зазначено вище синтаксичний аналізатор це дерево розбору коду, Kujira використовує алгоритм LALR. А алгоритм LALR такий: LALR відноситься до LR-lookahead. Для побудови таблиці розбору LALR (1) ми використовуємо канонічну колекцію елементів LR (1). У синтаксичному розборі LALR (1) елементи LR (1), які мають однакову продукцію, але мають інший погляд вперед, поєднуються для формування єдиного набору елементів. LALR (1) синтаксичний аналіз аналогічний розбору CLR (1), лише різниця в таблиці розбору[5].

Як можна помітити при парсінгу вихідний код перетворюється в структуру даних, а точніше в дерево. Чому в дерево? Тому що обробка такого дерева відбувається простіше і швидше.

Після всіх аналізів і парсинга вся структура переводиться в машинний код за допомогою C++. Поцес компіляції у C++ протикає у декілька етапів. Перший етап – попередня обробка(препроцесінг). Препроцесор – макро процесор, який перетворює програму для подалшого компілювання. На даній стадії відбувається робота з препроцесорними директивами(#include, #define, #ifndef, тощо.). Хедери, включені в програму за допомогою директиви #include, рекурсивно проходять стадію препроцесінга і включаються у випускаємий файл. Після з’являється файл який називається driver.ii. Другий етап – компіляція. Компіляція - перетворення отриманого на попередньому кроці код без директив в асемблерний код. Це проміжний крок між високорівневим мовою і машинним (бінарним) кодом. Третій етап – асемблювання. Асемблер перетворює асемблерний код в машинний код, зберігаючи його в об'єктному файлі. Об'єктний файл - це створений ассемблером проміжний файл, який зберігає шматок машинного коду. Цей шматок машинного коду, який ще не був пов'язаний разом з іншими шматками машинного коду в кінцеву виконувану програму, називається об'єктним кодом. Але на даному етапі ще нічого не закінчено, адже об'єктних файлів може бути багато і потрібно їх всіх з'єднати в єдиний виконуваний файл за допомогою компоновщика (линкера). Четвертий етап – компанування. Компоновщик (линкер) пов'язує всі об'єктні файли і статичні бібліотеки в єдиний виконуваний файл, який можна запускати. На рисунку 2.1 можна побачити більш детально процес перетворення коду в машинний.

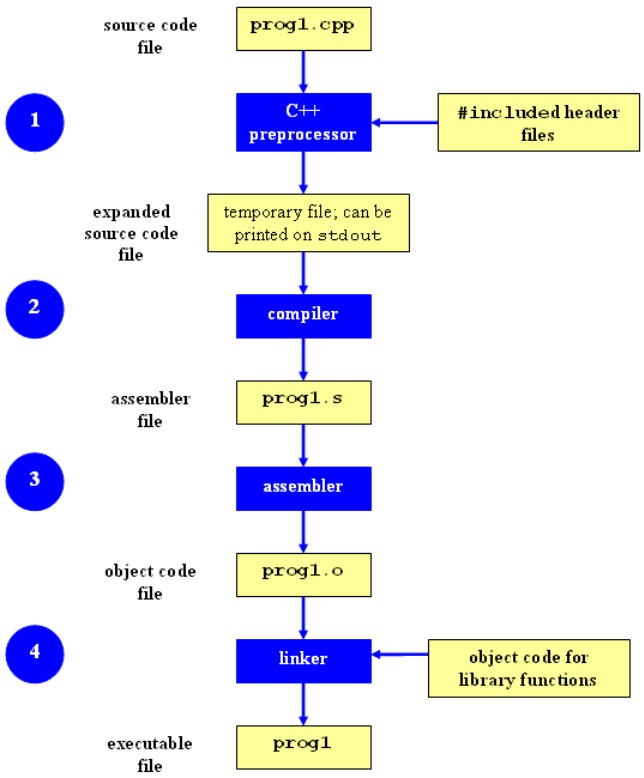


Рисунок 2.1 – Перетворення C++ коду в машинний код

Семантика мови програмування - це відповідність між синтаксично правильними програмами і діями абстрактного виконавця, тобто це сенс синтаксичних конструкцій. [2, 7] Розглянемо приклад з мови програмування Kujira.

Синтезовані атрибути - ідея, запропонована Дональдом Кнутом, полягала в тому, щоб зіставити кожному вузлу дерева розбору програми деяку функцію, що визначає семантику цього вузла. Подібна інформація зберігалася в так званих атрибутах. Семантика конструкції може представлятися деякою величиною або набором величин, пов'язаних з конструкцією. Наприклад, семантика виразу 3 + 4 може бути цілим значенням 7, типом int або рядком + 3 4. Величина, асоційована з конструкцією, називається атрибутом. Атрибут a для X будемо записувати як X.a, де X вважається не терміналом або терміналом граматики. E.val розглядається як посилання на атрибут val вираження E. [2]

Опис арифметичного правила використовуючи граматику LALR	Семантичне правило
?calc: calc "+" prod -> add	E.val := E.val1 + T.val
?calc: calc “-" prod -> sub	E.val := E.val1 - T.val
?calc: calc “*” prod -> mul	E.val := E.val1 * T.val

Так як дана мова відноситься до Python мови то переклад з природної мови в машинний відбувається за допомогою вбудованої віртуальної машини. Після реалізації граматики, лексеми і синтаксису можна протестувати мову. Для тестування коду користується кінцевий автомат. Кінцевий автомат - абстрактний автомат, число можливих внутрішніх станів якого звичайно.

2.4 Система типів та структура даних

У кожній мові програмування є свої типи даних, які виконують конкретні операції. Kujira також має свої типи даних.

Хоча Kujira - динамічно типізована мова програмування, вона все ж таки використовує прості типи.

Int - цілочисельний тип даних. Операції над цим типом завжди приводить до цілих чисел.

Float - метод float () повертає число з плаваючою комою з числа.

BigInt - цілочисельний тип даних використовується для операцій над числами більше ніж 2^{64}. Операції над цим типом завжди приводить до цілих чисел.

String - строковий літерал в Kujira оточені подвійними лапками.

Decimal - забезпечує підтримку арифметики з плаваючою комою з десятковою точкою. Він пропонує кілька переваг перед типом даних Float.

List - структура даних. Kujira як і Python не має масиву тому аналогом використовують список.

Set - для побудови та маніпулювання неупорядкованими колекціями унікальних елементів.

Dictionary - колекція, яка не упорядковується, змінюється та індексується. Словники у Kujira пишуться фігурними дужками, у них є ключі та значення. Аналогом може бути Hash.

На даний момент списки у Kujira не використовує різні типи даних, працює тільки з одним типом. Наприклад: `[1,2,3,4]` або `["a", "b", "c"]` теж саме з різними структурами даних окрім словника.

Бібліотека bioinfo використовує тип даних String або Int. Наприклад:

```
count_dna("ACTRGATCYGATCGANTCGATG")
```

Також у Kujira не передбачено створення своїх типів даних так як для цього треба створення ООП парадигми якої у Kujira немає.

2.5 Робот з сортуванням та порівняння с Python

Створивши деякі бібліотеки для мови програмування, було вирішено порівняти його з мовою програмування Python. Чому саме на ньому, тому що багато бібліотек написані на Python використовують C ++. Бібліотека також частково написана на C ++.

Спершу порівнюється сортування так як це найпростіше з чого можна почати і одне з найпопулярніших методів порівняння. Бібліотека була написана для сортування використовуючи стандартну STL бібліотеку де є вже sort (). Так як тестування буде проходити за допомогою мови Python то для власної функції sort () аргументами були вектора. Так як мова Python не має векторів, але має кортежі то використовуючи вектор фактично використовується кортежі Python'a.

Функція сортування:

```
std :: vector <int> arr (std :: vector <int> a) {...}
```

Наступним етапом було порівняння швидкості, так як C ++ є одним з найшвидших мов програмування, то можна було вважати, що C ++ по швидкості випередить Python. Але на подив швидким виявився Python, завдяки алгоритму сортування який він використовує, а використовує він Timsort. Timsort - це ефективна комбінація декількох алгоритмів, зокрема merge sort і insert sort. Відповідно функція sort Python була в рази швидше ніж C ++. Тоді я вирішив використовувати бібліотеку timsort C ++. Результати теж були цікаві.

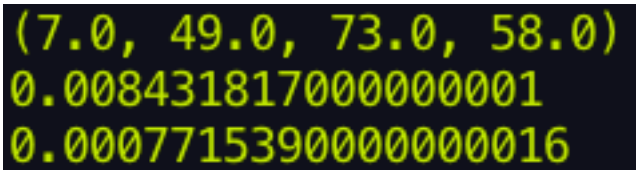


Рисунок 2.2 – Час сортування Python та Kujira

Перший час - час сортування написане на C ++, друге на Python. З чим це пов'язано? Це пов'язано з тим що у Python більшість модулів, а також ядро написано на мові C.

Робота з файлами:

Час читання невеликих файлів практично однакова. Але у Python'а є один недолік при читанні файлів - якщо файл містить багато даних.[19]

2.6 Висновки по розділу

У даному розділі було проведено огляд мови програмування, її граматика, лексика, семантика, типи та структури даних.

3 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура мови

Виконання програми виконується у декілька етапів лексичний аналіз, синтаксичний аналіз, семантичний та переклад у машинний код. Діаграма компонентів для розробки такого програмного забезпечення зображена на рисунку 3.1.

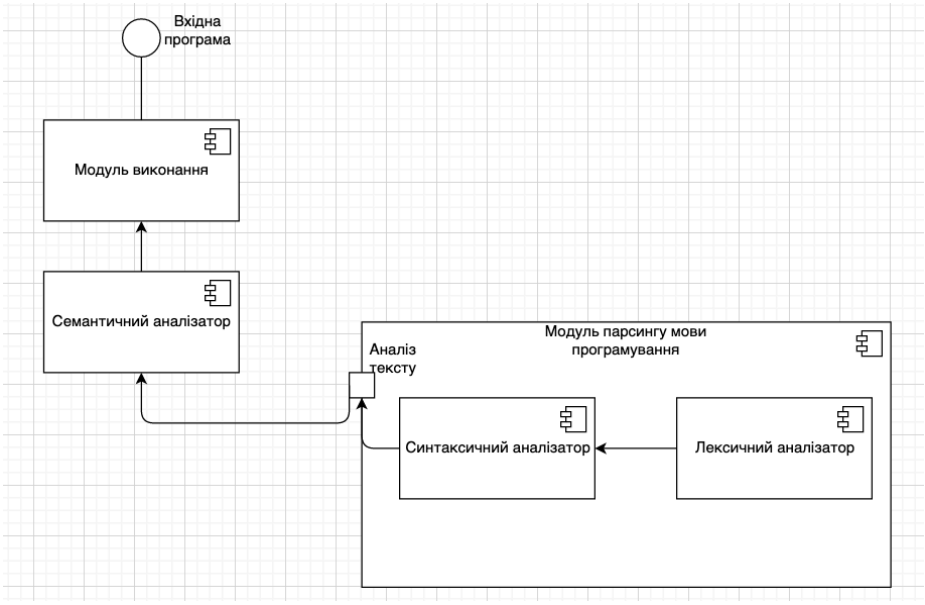


Рисунок 3.1 – Схема компонентів програми

Дана мова програмування, що інтерпретується, високорівнева і ставитися до гибрідної мови. Має гнучку динамічну типізацію даних схожу на такі мови як: Python, Ruby, Lisp або JavaScript. А також є Тьюринг-повним мовою, що означає має такі базові конструкції як if, while, for.

Дана мова так само як і Python не має точку входу, що означає, що можна прописувати будь-який код.

Для реалізації типів та бібліотек використовувався SWIG. SWIG - інструмент розробки програм, що автоматично генерує прив'язки між кодом C / C++ і поширеними мовами сценаріїв, включаючи Tcl, Python, Perl і Guile. SWIG підтримує більшість типів даних C / C++, включаючи покажчики, структури і

класи. На відміну від багатьох інших підходів, SWIG використовує декларації ANSI C / C++ і вимагає, щоб користувач практично не вносив змін до базового C-код.[8]

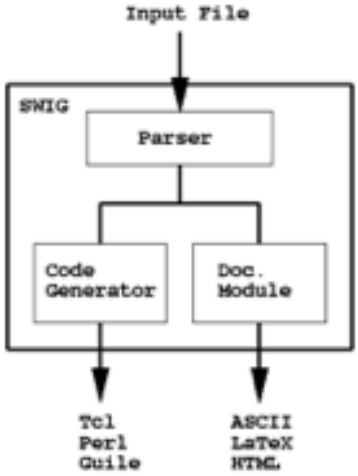


Рисунок 3.2- Архітектура SWIG

В основі SWIG лежить yacc для читання вхідного файлу і конвертації в відомі скриптові мови. Входять файлом в основному є код написаний на мові C / C++ який за допомогою yacc парсеру перекладається в один з даних мов Tcl, Python, Perl і Guile.

3.2 Архітектура мови(процес інтерпретації)

Кожна мова програмування в архітектурі має фронтенд, оптимізацію та бек-енд. Ці три стадії проходить будь-яка програма.

Kujira як і інші мови має цю архітектуру і на малюнку можна побачити.

Рисунок 3.3 - Інфраструктура Kujira

Але Kujira трохи відрізняється тим, що у Middle частині використовується SWIG. Тому мова у деяких етапах по швидкості може відставати від таких мов як C++ або Python. Процес інтерпретації реалізований у вигляді набору фаз, які викликаються для кожного блоку інтерпретації.

На першому етапі кожний вихідний файл аналізується в дереві програм. На наступних етапах вводяться символи в таблицю символів і анотується дерево програм з типами. Наступні фази знижуються, і виконують інші типові операції над деревами. Тобто інтерпретатор будує AST-дерево. Наприклад:

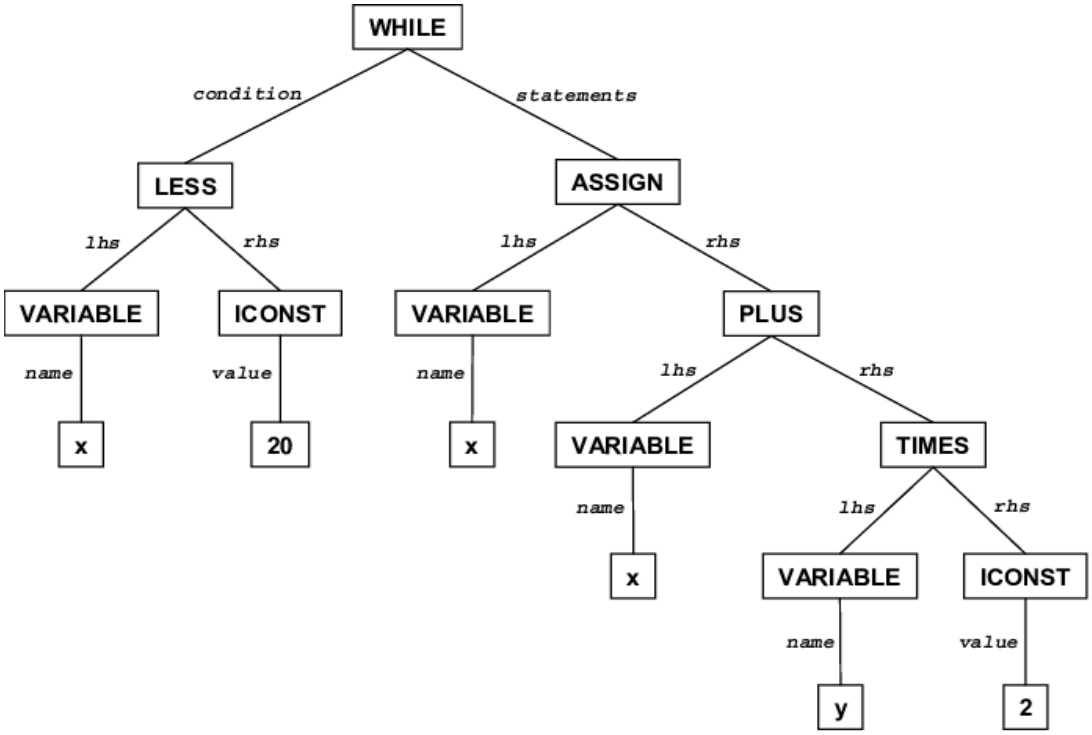


Рисунок 3.4 - AST-дерево

Один з пізніших етапів інтерпретації коли код для цільової платформи генерується безпосередньо з цих дерев. Для цього трансформери відвідують кожен вузол дерева і запускають на ньому відповідний метод відповідно до даних вузла, трансформер називається LanguageTransformer. На заключному етапі виводиться результат з LanguageTransformer.

```
    @v_args(inline=True)
    class LanguageTransformer(Transformer):
        from operator import add, sub, mul, truediv as
div, neg
        number = float
        def __init__(self):
            self.vars = {}
        def assign(self, name, args):
            self.vars[name] = args
            return args
        def var(self, name):
```

```
            return self.vars[name]
        def array(self, *elements):
            l = List.List()
            l.createnode(int(elements[0]))
            for arg in elements:
                l.insert_start(int(arg))
            l.display()
```

3.2 AST-дерево Kujira

Даний компонент відповідає за виконання генерованого та попередньо перевіреного семантично синтаксичного дерева. Мова програмування Kujira – має інтерпретатор, тому мова є інтрепретованою.

Оскільки прасинг вхідних текстів виконана за допомогою Python, хоча вбудовані функції, типи та структури даних написані на C++. Усі модулі компоненту виконання програми наведено та описано у таблиці 3.1

Таблиця 3.1 – опис модулів компоненту

Модуль	Призначення модулю
kujira.py	Використовується для парсингу вхідного файлу. Описує синтаксис мови за допомогою розширеної нотації Бекуса-Наура. Генерує початковий шаблон класів AST мовою програмування Python. Описує структури та типи даних. Описує клас для аналізу імпортованих модулей. Описує та відповідає за клас для перевірки синтаксичних помилок.

Продовження таблиці 3.1

importfile.py	Описує функцію для аналізу імпортованих модулей
module1.py	Один із прикладів модуля

Одним за найважливіших модулів є модуль kujira.py який генерує дерево та пов’язує згенероване дерево до C++ для подальшого обрахунку.

На мові C++ як було зазначено раніше створені типи та структури даних а також деякі вбудовані функції такі як математичні функції або сортування. У таблиці 3.2 наведені усі класи Kujira та їх опис та у таблиці 3.3 наведені усі класи Kujria та їх методи.

Таблиця 3.2 – основні класи типів та структур даних

Клас	Опис
Object	Клас Object є коренем ієрархії класів. У кожному класі Object є суперкласом. Усі об'єкти, включаючи список, реалізують методи цього класу.
Dictionary	Dictionary в Kujira - це упорядкований набір значень даних, який використовується для зберігання значень даних, як HashMap у Java або map у C++, яка на відміну від інших типів даних, що містить лише одне значення, як елемент, словник містить key: value значень. Ключ-значення надається у словнику, щоб зробити його більш оптимізованим.

Продовження таблиці 3.2

List	Найбільш основна структура даних в Python - це послідовність. Кожному елементу послідовності присвоюється число - його позиція або індекс. Список є найбільш універсальним типом даних, доступним у Kujira, який може бути записаний у вигляді списку розділених комами значень (елементів) між квадратними дужками. Важливим у списку є те, що елементи в списку повинні бути одного типу як у Java ArrayList. Також важливім є те, що список є однозв'язним.
Set	Set - це не упорядкований клас у Kujira. Кожен елемент набору є унікальним (без дублікатів) і повинен бути незмінним (неможливо змінити). Однак сам набір є змінним. Ми можемо додати або видалити з нього елементи. Набори також можуть бути використані для виконання математичних задач, таких як об'єднання, перетин, симетрична різниця тощо.

Продовження таблиці 3.2

Fraction	У Kujira модуль дробу підтримує раціональну арифметику чисел. Використовуючи цей модуль, ми можемо на даний момент створити дроби з цілих чисел. Існує концепція екземпляра дробу. Він утворений парою цілих чисел як чисельник і знаменник. Класи дробів. Fraction використовується для створення об’єкта Fraction. Знадобиться чисельник і знаменник.
Decimal	Decimal модуль забезпечує підтримку швидкої арифметики з десятковою плаваючою комою. Він пропонує кілька переваг перед типом даних Float. Decimal «заснована на моделі з плавоючою комою, яка була розроблена з урахуванням людей і обов'язково має найважливіший керівний принцип - комп’ютери повинні забезпечувати арифметику, яка працює так само, як арифметика, яку люди вивчають у школі». - уривок з десяткової специфікації арифметики.[17] Десяткові числа можуть бути

Продовження таблиці 3.2

	представлені точно. Навпаки, числа, такі як 1.1 та 2.2, не мають точного подання у двійковій плаваючій точці. Кінцеві користувачі зазвичай не очікують, що показник 1.1 + 2.2 відображатиметься як 3.3000000000000003. Отже Decimal відображає результат більш точно ніж Float.
Float	Клас Float зафіксує значення float примітивного типу в об'єкті. Об'єкт типу Float містить одне поле, тип якого є float.
Int	Клас Integer вказує значення int примітивного типу в об’єкт. Об'єкт типу Integer містить одне поле, тип якого є int. За замовчуванням тип даних int - це 32-розрядне ціле число доповнення двох, яке має мінімальне значення -2^{31} та максимальне значення $2^{31}-1$. У Kujira можна використовувати тип даних int для представлення непідписаного 32-бітного цілого числа, яке має мінімальне значення 0 і максимальне значення $2^{31}-1$. Використовуйте клас Integer, щоб

Продовження таблиці 3.2

	використовувати тип даних int як ціле число без підпису.
String	Клас String представляє символьні рядки. Усі рядкові літерали в програмах Kujira, такі як "abc", реалізовані як екземпляри цього класу. String являє собою рядок у форматі UTF-8.
BigInt	Клас BigInt використовується для математичної операції, яка включає дуже великі цілі обчислення, які виходять за межі всіх доступних примітивних типів даних. Наприклад, факторіал від 100 буде містити 158 цифр, тому ми не можемо зберігати їх у будь-якому наявному примітивному типі даних. Ми можемо зберігати в ньому такий великий Integer, який ми хочемо. На верхній межі діапазону немає теоретичного обмеження, оскільки пам'ять розподіляється динамічно.

Таблиця 3.3 – основні методи типів та структур даних

Клас	Назва методу	Опис
Object	getClass()	Повертає клас виконання цього об’єкта.

Продовження таблиці 3.3

	equals(Object& obj)	Вказує, чи є якийсь інший об'єкт "рівним" цьому.
	clone()	Створює та повертає копію цього об’єкта.
	toString()	Повертає рядкове подання об'єкта.
Dictionary	print_map(map<K,V> const &m)	Виведення результату
	ensureCapacity()	збільшення розміру ємності Dictionary
	Dictionary(int s)	Побудує порожній Dictionary із заданою початковою ємністю
	set(K elem, V val)	Пов’язує вказане значення із вказаним ключем на цій карті.
	get()	Повертає значення, для якого вказаний вказаний ключ, або null, якщо ця карта не містить відображення ключа.

Продовження таблиці 3.3

	~Dictionary()	destructor(~Dictionary) - метод, який автоматично викликається при знищенні об'єкта
List	createnode(int value)	Створення вузла списку
	display()	Вивести список
	insert_start(int value)	Додати перше занчення у список
	insert_position(int pos, int value)	Додати занчення у позицію
	delete_first()	Видалення першого елементу з списку
	delete_last()	Видалення останнього елементу з списку
	delete_position(int pos)	Видалення елементу зі списку за конкретною позицією
	find_elem(int data)	Знаходження елементу та виведення її позиції у списку
	size()	Отримання розміру списку
	clear()	Видалення всіх існуючих елементів списку

Продовження таблиці 3.3

Set	insert(Node *n, int x)	Додати перший елемент у Set
	search(int x, Node *n)	Пошук елементу у множині
	min(Node *n)	Знаходження мінімального елемента у множині
	max(Node *n)	Знаходження максимального елементу у множині
	inorder(Node *n)	Знаходження позиції елемента у множині
	put(int x)	Додавання нового елементу у множину
	find(int x)	Знаходження елементу у множині
	max_elem(Node *n)	Знаходження максимального елементу у множині
	min_elem(Node *n)	Знаходження мінімального елемента у множині
	display()	Виведення множини на екран
Fraction	Fraction()	Перший конструктор є конструктором без параметру

Продовження таблиці 3.3

	Fraction(int num, int den)	Другий конструктор вимагає, щоб чисельник і знаменник були екземплярами чисел. Повертає новий екземпляр дробу зі значенням чисельник / знаменник.
	getNumerator()	Чисельник дробу в найнижчому часі
	getDenominator()	Знаменник дробу в найнижчому терміні.
	operator+(Fraction otherFrac)	Функція для обчислення суми двох дробів
	operator-(Fraction otherFrac)	Функція для обчислення віднімання двох дробів
	operator*(Fraction otherFrac)	Функція для обчислення добутку двох дробів
	operator/(Fraction otherFrac)	Функція для обчислення ділення двох дробів

Продовження таблиці 3.3

	toString()	Повертає рядкове подання об'єкта.
Decimal	Decimal(string number)	У конструктор передається строкове значення, яке представляє число
	operator+(Decimal &number, Decimal &number2)	Функція для обчислення суми двох десяткових чисел
	operator-(Decimal &number, Decimal &number2)	Функція для обчислення віднімання двох десяткових чисел
	operator*(Decimal &number, Decimal &number2)	Функція для обчислення добутку двох десяткових чисел
	operator/(Decimal &number, Decimal &number2)	Функція для обчислення ділення двох десяткових чисел
	value()	Отримання значення яке є у конструкторі або після арифметичної операції

Продовження таблиці 3.3

Float	Float(float number)	Побудує щойно виділений об'єкт Float, який представляє примітивний аргумент float.
	operator+(Float &number, Float &number2)	Функція для обчислення суми двох float чисел
	operator-(Float &number, Float &number2)	Функція для обчислення віднімання двох float чисел
	operator*(Float &number, Float &number2)	Функція для обчислення добутку двох float чисел
	operator/(Float &number, Float &number2)	Функція для обчислення ділення двох float чисел
	ostream &operator<<(ostream &out, Float &num)	Функція для виведення на екран float
	value()	Повертає float значення цього об'єкта Float.
	__hash__(const Float &number)	Повертає хеш-код для цього об'єкту Float.

Продовження таблиці 3.3

Int	Int(int number)	Побудує щойно виділений об'єкт Integer, який представляє вказане значення int.
	Int operator+(Int &number, Int &number2)	Функція для обчислення суми двох int чисел
	Int operator-(Int &number, Int &number2)	Функція для обчислення віднімання двох int чисел
	Int operator*(Int &number, Int &number2)	Функція для обчислення добутку двох int чисел
	Int operator/(Int &number, Int &number2)	Функція для обчислення ділення двох int чисел
	ostream &operator<<(ostream &out, Int &num)	Функція для виведення на екран int числу
	value()	Повертає значення цього цілого числа як int.
	__hash__(const Int &number)	Повертає хеш-код для цього об'єкту Int.

Продовження таблиці 3.3

String	String(string str)	Ініціалізує новостворений об'єкт String, щоб він представляв ту саму послідовність символів, що і аргумент; іншими словами, новостворена рядок є копією рядка аргументу.
	operator+(String& str1, String& str2)	Функція для конкатенації двох String значень
	size()	Повертає довжину цього рядка.
	capitalize()	Перетворює перший символ рядка в велику (велику) букву. Якщо рядок має перший символ у якості капіталу, він повертає початковий рядок.
	upper()	Перетворює всі символи цього рядка у верхній регістр, використовуючи правила локалі за замовчуванням.

Продовження таблиці 3.3

	lower()	Перетворює всі символи цього рядка в малі регістри, використовуючи правила локальної програми за замовчуванням.
	value()	Повертає рядкове подання аргументу string.
BigInt	BigInt()	Перший конструктор є конструктором без параметру
	BigInt(string num1, string num2)	Перетворює десяткове представлення рядка BigInt в BigInt.
	multiply(string num1, string num2)	Повертає BigInteger, значення якого (this * val).
	addition(string num1, string num2)	Повертає BigInteger, значення якого (this + val).
	substraction(string num1, string num2)	Повертає BigInteger, значення якого (this - val).

Одним із найважливіших для парсиінгу вхідного тексту є AST-дерево. Воно створюється на стадії синтаксичного розбору, обробляється шляхом обходу при перевірці семантичних правил і перевірці / визначенні типів, а потім також шляхом обходу AST виконується генерація коду. Для цього використовувався lark. Lark – це бібліотека для створення дерева та розробки синтаксису майбутньої мови. Використовує розширену нотацію Бекуса-Наура. У таблиці 3.4 можна побачити усі класи бібліотеки lark.

Таблиця 3.4 – опис класів бібліотеки lark.

Клас	Опис класу	Назва методу	Опис методу
Lark	Головний клас для конструювання дерев та для створення граматики	__init__(self, grammar_string, **options)	Створює об’єкт класу Lark із заданою граматикою
		open(cls, grammar_filename, rel_to=None, **options)	Створює екземпляр Lark з граматикою, заданою його іменем файлу. Якщо надано rel_to, функція знайде ім'я файлу граматики стосовно нього.
		parse(self, text)	Повертає повне дерево розбору для тексту

Продовження таблиці 3.4

		save(self, f) / load(cls, f)	Корисно для кешування та багатопроцесорної обробки. save зберігає екземпляр у заданому файловому об’єкті load завантажує екземпляр із заданого файлового об'єкта
Tree	Клас бібліотеки lark для створення дерева AST	__init__(self, data, children)	Створює нове дерево та зберігає "дані" та "діти" в однойменних атрибутах.
		pretty(self, indent_str=' ')	Повертає з відступом рядкове зображення дерева. Використовується для налагодження.

Продовження таблиці 3.4

		find_pred(self, pred)	Повертає всі вузли дерева, які оцінюють pred (вузол) тобто попередній як істинний.
		find_data(self, data)	Повертає всі вузли дерева, дані яких дорівнюють заданим даним.
		iter_subtrees(self)	Пошук в глубину з ітеративним заглибленням. Вступає в взаємодію над усіма підрядками, ніколи не повертаючись до одного і того ж вузла двічі.

Продовження таблиці 3.4

		iter_subtrees_topdown(self)	Пошук у ширину. Ітераціює над усіма підрядчиками, повертає вузли в порядку, як досить ().
Token	При використанні лексеру отримані лексеми в деревах будуть класом Token, який успадковується з рядка Python. Отже, звичайні порівняння рядків і операції працюватимуть, як очікувалося.	__new__(cls, type_, value, pos_in_stream=None, line=None, column=None, end_line=None, end_column=None)	Метод __new__ викликається автоматично при виклику імені класу. Параметри методу: type - Назва маркера (як зазначено в граматиці). pos_in_stream - індекс маркера в тексті рядок - рядок маркера в тексті (починаючи з 1) стовпець - стовпець маркера в тексті (починаючи з 1)

Продовження таблиці 3.4

			end_line - рядок, де маркер закінчується
			end_column – наступний стовпець після закінчення маркера. Наприклад, якщо маркер є одним символом зі значенням стовпця 4, кінець стовпця буде end_pos – індекс, на якому закінчується маркер (в основному pos_in_stream + len(маркер))
		__reduce__(self)	Викликається коли серіалізуються об'єкт, в якому цей метод був визначений.

Продовження таблиці 3.4

		__repr__(self)	Викликається вбудованою функцією repr; повертає "сирі" дані, що використовуються для внутрішнього представлення в Python.
		__deepcopy__(self, memo)	Глубоке копіювання об’єкту
Visitors	Visitors відвідує кожен вузол дерева та запускає на ньому відповідний метод відповідно до даних вузла. Працює знизу вгору, починаючи з листя і закінчуючи біля кореня дерева. Тобто використовує алгоритм LL	visit(self, tree)	

Продовження таблиці 3.4

		visit_topdown(self,tree)	
Transformer	Відвідує дерево рекурсивно, починаючи з листя і, нарешті, корінь (знизу вгору) Викликає його методи (надаються користувачем через спадщину) відповідно до tree.data Повернене значення замінює старе в структурі. Може використовуватися для реалізації карти або зменшення.	__init__(self, visit_tokens=True)	
		_call_userfunc(self, tree, new_children=None)	
		_call_userfunc_token(self, token)	

Продовження таблиці 3.4

		_transform_children(self, children)	Відвідування дерева рекурсивно та отримання вершини дерва
		_transform_tree(self, tree)	Відвідування дерева рекурсивно
		transform(self, tree)	Відвідування дерева рекурсивно
		__mul__(self, other)	Множення (x*y)
		__default__(self, data, children, meta)	Операція за замовчуванням на дереві (для перевизначення)
		__default_token__(self, token)	Операція за замовчуванням у токені (для перевизначення)

3.3 Алгоритми Kujira

З розвитком методів біохімії та молекулярної біології з одного боку, і обчислювальної техніки - з іншого, в біологічній науці сформувався особливий підхід до вивчення живих систем. Даний підхід передбачає дослідження біологічних систем із застосуванням обчислювальних машин, тобто розглядає

живі системи як набір інформації про них. Однією з біологічних наук є біоінформатика.

Предметом біоінформатики є алгоритми обробки даних про структуру біологічних макромолекул. Об'єкт біоінформатики - нуклеїнові кислоти і білки, біологічні макромолекули, чия структура принципово визначає властивості живих організмів і їх частин.

Робота з первинною структурою біологічних макромолекул - тобто з нуклеотидної або амінокислотної послідовністю - є, мабуть, найбільш поширеною завданням біоінформатики. Цьому сприяє, насамперед, відносна легкість отримання інформації про первинну структуру нуклеїнових кислот і білків на сучасному етапі розвитку біохімії і молекулярної біології. На порівнянні послідовностей нуклеїнових кислот засновані сучасні методи класифікації живих організмів, а аналіз безлічі гомологічних білкових послідовностей дозволяє отримати інформацію про структурні і функціональні особливості молекули.

Послідовності білків і нуклеїнових кислот дуже легко можуть бути записані у вигляді тексту, оскільки і ті, і інші є лінійними, тобто не розгалужуються.[18]

Символ	Обозначаемые остатки	Мнемоническое правило
A	A	**A**denine
C	C	**C**ytosine
G	G	**G**uanine
T	T	**T**hymine
U	U	**U**racil
R	A или G	pu**R**ine
Y	C, T или U	p**Y**rimidines
K	G, T или U	**K**етон
M	A или C	Содержит a**M**иногруппу
S	C или G	**S**trong interaction (три водородные связи)
W	A, T или U	**W**eak interaction (две водородные связи)
B	Любой, кроме A	после A
D	Любой, кроме C	после C
H	Любой, кроме G	после G
V	Любой, кроме T и U	после U
N	Любой	**N**ucleic acid
X	Любой	Неизвестная величина

Рисунок 3.5-Однолітерні позначення нуклеотидних залишків

Для запису біологічних послідовностей використовується формат Fasta - текст, розмічений особливим чином. Кожна послідовність може бути випереджу заголовком, що містить знак ">" і назва послідовності. Тема займає окремий рядок; після розриву рядків записується послідовність.[18]

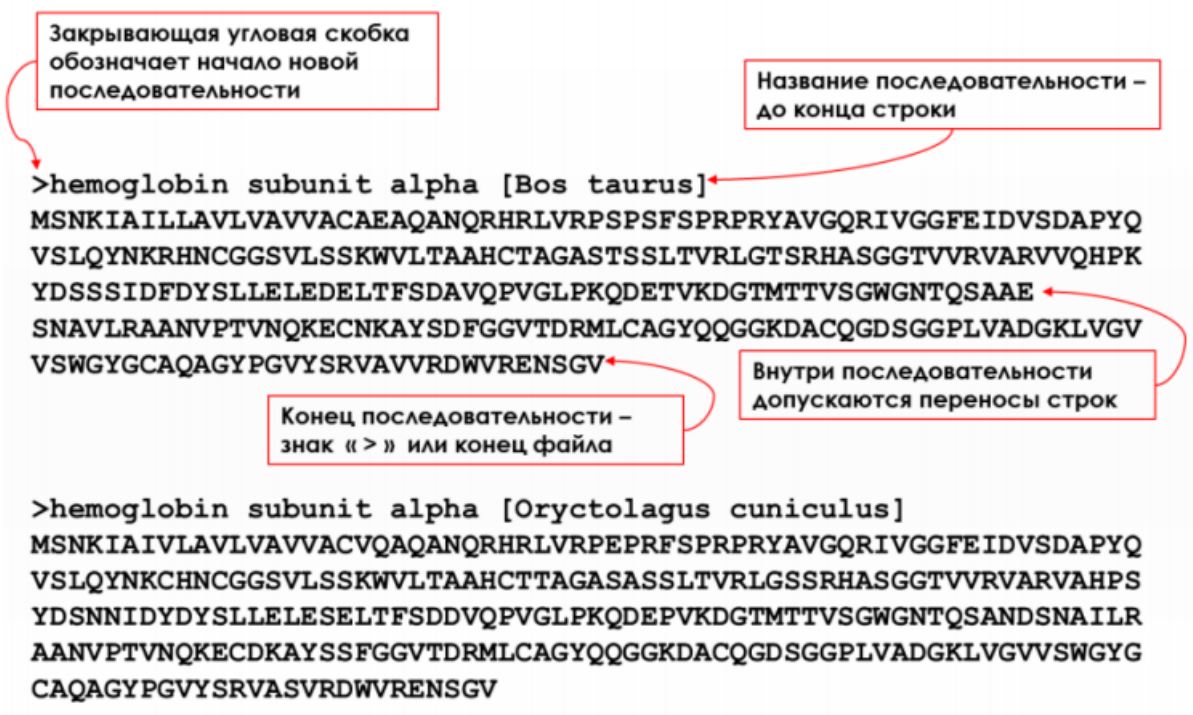


Рисунок 3.6-Fasta-послідовність

3.4 Висновки по розділу

У рамках даного розділу було оглянуто архітектуру мови програмування її складові, а також алгоритм парсингу та приклад створення AST-дерева.

4 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості

Будь-який програмний продукт повинен виконувати ті функції, для яких був створений. Якісний продукт повинен володіти ще рядом властивостей, що дозволяють успішно його використовувати протягом тривалого часу. Також у програмного продукту є критерії за якими перевіряється якість.

Критерії програмного продукту:

– функціональність;

– надійність;

– легкість застосування;

– ефективність;

– мобільність;

– безпека.

4.1.1 Функціональність

Функціональність - це здатність програмного продукту виконувати набір функцій, визначених його зовнішніми специфікаціями.

Функції програмного продукту:

- мова надає можливість писати програми;
- мова надає можливість використовувати вбудовану бібліотеку bioinfo та використати її у біоінформатиці;
- мова надає можливість створювати власноруч створені бібліотеки на мові Kujira або Python;
- система повинна надавати функції аналізатора коду, які можуть приймати весь код і розділитись на вирази;

система повинна містити помилки або винятки, які містяться в коді.

4.1.2 Надійність

Надійність - це здатність безвідмовно виконувати задані функції при заданих умовах протягом заданого періоду часу з високим ступенем ймовірності.

Система повинна використовувати основну мову Python 3 для запуску, тому вона не може бути більш надійною, ніж сама Python 3. Сказавши це, якщо розробник використовує власні файли, написані на Kujira, і генеруються помилки, всі помилки повинні бути передані розробнику, де їм дозволено належним чином обробляти їх. Наприклад, синтаксичні помилки. Синтаксичні помилки - це основний тип помилок. Вони виникають, коли аналізатор Kujira не в змозі зрозуміти рядок коду. Помилки синтаксису майже завжди є фатальними, тобто майже ніколи немає способу успішного виконання фрагмента коду, що містить синтаксичні помилки.

Логічні помилки - це найскладніший тип помилок, оскільки вони дадуть непередбачувані результати та можуть збійнути програму користувача.

4.1.3 Легкість застосування

Легкість застосування - це здатність мінімізувати втрати користувача на підготовку до використання.

Kujira має схожий синтаксис з Python тому легкість розуміння буде вища ніж розуміння таких мов як C++ та R.

4.1.4 Ефективність

Ефективність - це здатність обробляти дані достатньо швидко.

Усі модулі написані на C++ та частково на Python. Порівнюючи швидкість сортування Python та Kujira, можна побачити, що Kujira достатньо швидка мова.

4.1.5 Мобільність

Мобільність - це здатність програмного продукту бути перенесеним з одного обчислювального середовища в іншу.

- Система повинна мати можливість працювати в системах Windows, де встановлено Python 3.
- Система повинна мати можливість працювати в системах Macintosh, де встановлено Python 3.
- Система повинна мати можливість працювати в системах Unix, де встановлено Python 3.
- Код повинен бути читабельним, добре коментованим та підтримуваним.
- Система повинна бути записана на Python та C / C ++.

4.1.6 Безпека

Хоча система не є критичною і не використовуватиметься для підключення до мережі, все одно важливо захистити користувача від локальних атак. Весь вхід буде перевірено, щоб переконатися, що він не спричинить переповнення буфера або не додасть до нестабільності системи. Також використовується автоматичний garbage collection від Python тому проблеми з переповненням буфера або шось подібного не відбудеться.

4.2 Опис процесів тестування

Для перевірки відповідності функціональним вимогам була проведена черга тестів, які перевірили коректність роботи мови. У кожної вимоги проводилось декілька тестів, які дозволяють гарантувати відповідність розробленої мови програмування.

Вхідними даними були написані маленькі скрипти за допомогою мови програмування Kujira. Також були проведені ряд тестувань на коректність роботи бібліотеки bioinfo.

4.3 Опис контрольних прикладів

4.3.1 Тестування мови програмування

У рамках даного пункту усі контрольні перевірки виконуються в однакових умовах: створене одне середовище, вихідний текст програми подається та інтерпретатор як вхідна дія.

Таблиця 4.1 - Обробка помилок

Номер	№1
Мета тесту	Перевірка синтаксичної помилки (відсутність дужки)
Вхідна програма	a = [1, 2, 3, 4
Виконання тесту	Запуск коду
Очікуваний результат	Виведення повідомлення про помилку
Фактичний результат	Програма не завершилася успішно, виведена помилка: [1,2,3,4 ^ at line 1, column 8.
Номер	№2
Мета тесту	Перевірка обробки лексичної помилки (одинарні лапки замість подвійних)
Вхідна програма	print ‘Hello’

Продовження таблиці 4.1

Виконання тесту	Запуск коду
Очікуваний результат	Виведення повідомлення про помилку
Фактичний результат	Програма не завершилася успішно, виведена помилка: print 'Hello' ^ at line 1, column 7.
Номер	№3
Мета тесту	Застосування невідповідного типу до об'єкта
Вхідна програма	sum = ‘12’ + 34
Виконання тесту	Запуск коду
Очікуваний результат	Виведення повідомлення про помилку
Фактичний результат	Програма не завершилася успішно, виведена помилка: sum = '12' + 34 ^ at line 1, column 7.
Номер	№4

Продовження таблиці 4.1

Мета тесту	Об’єкт не вдалося знайти або не проініціалізован
Вхідна програма	http_pars=
Виконання тусту	Запуск програми
Очіуваний результата	Виведення повідомлення про помилку
Фактичний результат	Програма не завершилася успішно, виведена помилка: http_pars= ^ at line 1, column 4.
Номер	№5
Мета тесту	Перевірка синтаксичної помилки (if-then-else)
Вхідна програма	iff 12 < 13 then print “true” else print “false”
Виконання тусту	Запуск програми
Очіуваний результата	Виведення повідомлення про помилку

Продовження таблиці 4.1

Фактичний результат	Програма не завершилася успішно, виведена помилка: iff 12 < 12 then print "true" else print "fa ^ at line 1 col 5
Номер	№6
Мета тесту	Перевірка вкладених функцій

Вхідна програма	void a() {
	void printH() {
	print “Hello”
	}
	}

Виконання тусту	Запуск програми
Очіуваний резульата	Виведення повідомлення про помилку
Фактичний результат	Програма не завершилася успішно, виведена помилка: a() { void printH() { print "Hello" } } printH() ^ at line 1 col 46
Номер	№7

Продовження таблиці 4.1

Мета тесту	Список з різними типами даних
Вхідна програма	li = [1, 2, 3, “a”, “b”, “c”]
Виконання тусту	Запуск програми
Очіуваний резульата	Виведення повідомлення про помилку
Фактичний результат	Програма не завершилася успішно, виведена помилка: li = [1, 2, 3, "a"] ^ at line 1 col 14
Номер	№8
Мета тесту	Імпортування модулів
Вхідна програма	import mathlib cos_ 12
Виконання тусту	Запуск програми
Очіуваний резульата	Виведення повідомлення про помилку
Фактичний результат	Програма не завершилася успішно, виведена помилка: importtt mathlib cos_ 12 ^ at line 1 col 10

4.4 Висновки по розділу

У даному розділі було проведено ряд перевірок розробленого інтерпретатору. За результатами тестування можна сказати, що розроблений продукт відповідає функціональним вимогам та описаній семантиці та синтаксису мови Kujira.

5 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Розгортання програмного забезпечення для запуску програми написаних на мові Kujira, складається у розгортанні інтерпретатора. Також треба зазначити, що розробка мови та тестування велися у операційній системі MacOS 10.14, хоча мова може використовуватись у інших системах таких як Windows або Linux-дистрибутивах але для роботи треба мати встановлену Python версії 3.

Сама Kujira написана на двох мовах Python та C++. Також використовується бібліотека lark для роботи з граматикою та синтаксисом мови.

Компоненти синтаксичного та лексичного аналізу вимагають теж саме встановлення lark бібліотеки і SWIG адже багато компонентів були написані саме на C++. Необхідні файли для встановлення зазначені у файлі requirements.md та можуть бути встановлені за допомогою пакетного менеджера pip версії 3. Більш детальніше про встановлення та налаштування можна дізнатися у файлі installation.md.

5.2 Робота з мовою програмування

Опис граматики мови наведено у розділі 2 “Опис розробленої мови програмування”. Детальніше про налаштування та запуск можна дізнатись у додатку “Інструкція користувача”.

5.3 Супровід програмного

Супровід програмного забезпечення перш за все передбачає написання детальної документації та слідкування за неї. А також оптимізацією деяких деталей у мові для покращення стану роботи.

5.4 Висновки по розділу

У розділ були розглянуті необхідні інструменти та сторонні бібліотеки, які використовуються для розгортання розробленого програмного забезпечення та його запуску. Також було оглянуто необхідні заходи для супровіду програми під час виконання та розроблено інструкцію, поміщену у окремий додаток.

ВИСНОВКИ

Протягом роботи над даним дипломним проектом було розглянуті алгоритми які використовуються у біоінформатиці. Був проведений аналіз предметної області та оглянуті вже існуючі бібліотеки які найчастіше використовуються у мовах програмування. На основі цих досліджень були розроблені функціональні вимоги до програмного забезпечення та описано мову програмування, яка гарантує забезпечення цих вимог.

Для описаної мови було прораховано архітектуру. На основі реалізованої архітектури було прийняті рішення щодо технологій, які слід використовувати. Розроблений інтерпретатор та інструменти для розв’язання задач з біоінформатики були ретельно протестовані на відповідність вимогам та продемонстровано у інструкції користувача. Результати роботи показує можливості використання мови для написання програм не тільки з біоінформатики але й для програм загального призначення.

У розділі “Впровадження та супровід програмного забезпечення” було наведено основні вимоги до запуску мови програмування Kujira.

Перспективою розвитку даної мови та інструментів для неї перш за все мають стати розширення системи типізації для біоінформатики, а також нові функції та структури даних для обчислення задач з біоінформатики та прискорення процесу обробки даних. Систему типізації має сенс розширювати для відповідності сучасним об’єктно-орієнтованим мовам програмування для забезпечення кращої надійності.

ПЕРЕЛІК ПОСИЛАНЬ

1. Синтаксис, семантика и прагматика [Онлайн]. Доступно: https://foxford.ru/wiki/informatika/sintaksis-semantika-i-pragmatika
2. Формальная семантика языков программирования [Онлайн]. Доступно: http://storage.piter.com/upload/contents/978549600032/978549600032_p.pdf
3. LALR parser[Онлайн]. Доступно: https://en.wikipedia.org/wiki/LALR_parser
4. Amin Milani Fard, Arash Deldari and Hossein Deldari, Quick Grammar Type Recognition: Concepts and Techniques – 2007 https://www.researchgate.net/publication/228609916_Quick_Grammar_Type_Recognition_Concepts_and_Techniques
5. Frank DeRemer, Thomas J. Pennello, Efficient computation of LALR(1) look-ahead sets // ACM SIGPLAN Notices. – 1979. Volume – 14, Issue 8 Доступно: https://dl.acm.org/citation.cfm?id=800229.806968
6. Maggie Johnson, LALR Parsing. Доступно: https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/140%20LALR%20Parsing.pdf
7. Орлов С.А. Теория и практика языков программирования: Учебник для вузов. Стандарт 3-го поколения**,** 2013, с.182.
8. David M. Beazley, SWIG : An Easy to Use Tool for Integrating Scripting Languages with C and C++ // Presented at the 4th Annual Tcl/Tk Workshop - 1996.
9. lark[Онлайн]. Доступно: https://github.com/lark-parser/lark.
10. IEEE Recommended Practice for Architectural Description of Software intensive Systems. Standard 1471-2000, IEEE, 2000.
11. Object Management Group. Unified Modeling Language: Superstructure, Version 2.0.Technical report, OMG, 2004.
12. https://julialang.org/blog/2016/04/biojulia2016/

13. Hubert Baumeister, Florian Hacklinger, Rolf Hennicker, Alexander Knapp, Martin Wirsing //A Component Model for Architectural Programming, Electronic Notes in Theoretical Computer Science Volume 160, 2006, с.75-96.
14. Programming languages and the biological sciences https://dl.acm.org/doi/pdf/10.5555/1229637.1229669
15. Biopython Tutorial and Cookbook http://biopython.org/DIST/docs/tutorial/Tutorial.pdf
16. Illustrating Python via Bioinformatics Examples http://hplgit.github.io/bioinf-py/doc/pub/bioinf-py.html#bioinf:oo
17. https://docs.python.org/3/library/decimal.html
18. https://биоуфа.рф/%D0%B1%D0%B8%D0%BE%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0/#1
19. Очеретяний О.К., Каджая В.М., Баклан І.В., Конференція з АУ-2020, ст.49
20. Каджая В.М., Архітектура мови програмування Kujira, ІСТУ-2020, ст.171
21. Каджая В.М., Баклан І.В. Огляд мови програмування KUJIRA, Комп’ютерно-інтегровані технології у сьогоденні 3, ст.19

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ПРОБЛЕМНО-ОРІЄНТОВНА МОВА ДЛЯ ПРОГРАМУВАННЯ ЗАДАЧ З БІОІНФОРМАТИКИ

Технічне завдання

КПІ.ІП-6111.045490.02.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ І.В. Баклан

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____В.М. Каджая

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Проблемно-орієнтовна мова для програмування задач з біоінформатики

Галузь застосування: написання програмістами або вченими програми для роботи з задачами із біоінформатики

Наведене технічне завдання поширюється на розробку мови програмування Kujira, котра використовується для обчислення біоінформаційних задач та призначена для використання у біоінформатики.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки мови програмування Kujira є завдання на дипломне проектування, затверджено кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут» (НТУУ «КПІ»).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання мови програмування Kujira у біоінформатиці.

Метою розробки є проектування мови програмування для моделювання експериментів із злиття клітин.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

- написання та компіляція програми користувача;
- імпортування зовнішніх бібліотек;
- знаходження синтаксичних помилок користувача та повідомлення про них;

4.1.2 Вимоги до організації вхідних даних

4.1.3 Вимоги до організації вихідних даних

4.1.4 Розробку виконати на платформі Mac OSX

4.2 Вимоги до надійності

Система використовує Python для парсингу, а C++ для оптимізації, тому вона не може бути більш надійною, ніж самі ці мови. Сказавши це, якщо розробник використовує власні файли, написані на мові Kujira, і генеруються помилки, всі помилки повинні бути передані розробнику, де вони можуть обробляти їх відповідним чином. Наприклад, синтаксичні помилки. Синтаксичні помилки - найголовніший тип помилок. Вони виникають, коли парсер Kujira не може зрозуміти рядок коду. Синтаксичні помилки майже завжди фатальні, тобто майже ніколи не існує способу успішно виконати фрагмент коду, що містить синтаксичні помилки.

Логічні помилки - найскладніший тип помилок, які важко можна знайти, тому що вони дають непередбачувані результати і можуть призвести до збою користувальницької програми.

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних.

Не висовуються

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96

Не висуваються.

4.3.2 Обслуговування

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1 Тип процесору

Intel або ARM

4.4.2.2 Об‘єм ОЗП

8 Гб.

4.5 Вимоги до інформаційної та програмної сумісності

а) Програмне забезпечення повинно працювати під управлінням операційних систем сімейства сімейства Windows або Unix.

б) Вхідні дані повинні бути представлені в наступному форматі: код написаний на мові Kujira.

в) Результати повинні бути представлені в наступному форматі: кінцеві значення отримані з виконання коду або список помилок.

Мова програмування Kujira використовує дві мови Python для парсингу та C++ для оптимізації. Середовищем розробки було Visual Studio Code. Також використовувався фреймворк SWIG який є проміжним етапом після парсингу та до оптимізації. В якості захисту використовується синтаксичний аналізатор який виявляє помилки та інформує користувача.

4.5.1 Контекстна граматика

Контекстна граматика складається з ряду постановки. Кожна постановка має абстрактний символ, званий нетерміналом як його лівого боку, і послідовність одного або більше нетермінальних і термінальних символів в якості його правій частині. Для кожної діаграми термінальні символи взяті з заданого алфавіту. Починаючи з пропозиції, що складається з одного виділеного нетермінала, званого символом мети, дана не залежить від контексту граматика задає мову, а саме набір можливих послідовностей термінальних символів, які можуть бути отримані від багаторазового заміни будь-якого нетермінала в послідовності на праву сторону виробництва, для якого нетермінал є лівою стороною.

4.5.2 Граматика

Лексична граматика для мови програмування Kujira. Ця граматика має в якості термінальних символів символи набору символів Unicode. Він визначає набір постановки, починаючи з вхідного цільового символу , який описує, як послідовності символів Unicode переводяться в послідовність елементів введення . Ці елементи введення з пропуском і коментарі відкидаються, утворюючи термінальні символи для синтаксичної граматики мови програмування Kujira які називаються токенами. Цими токенами є ідентифікатори, ключові слова, літерали, роздільники і оператори мови програмування Kujira.

4.5.3 Нотація граматики

Термінальні символи показані шрифтом фіксованої ширини в похідних лексичної і синтаксичної граматики і у всій цій специфікації, коли текст прямо посилається на такий термінальний символ. Вони повинні відображатися в програмі точно так, як написано. Необмежені символи показані курсивом. Визначення нетермінала вводиться ім'ям визначається нетермінала, за яким слідує двокрапка. Потім слід одне або кілька альтернативних визначень нетермінала.

Приклад:

ifThenStatement:

if (Expression) Statement

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати установочну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

а) Супроводжувальна документація

1) Пояснювальна записка.

2) Керівництво користувача.

3) Керівництво системного програміста.

4) Керівництво адміністратора.

5) Програма та методика тестування.

б) Довідникова документація

1) Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

2) Програмне забезпечення повинно мати вбудовану довідку.

в) Графічна документація

1) Схема структура інформаційної системи.

2) Схема структурная программного обеспечения.

3) Схема функціональна програмного забезпечення.

4) Схема структура потоків даних програмного забезпечення або його частини.

5) Схема структурна компонентів структур даних.

6) Схема структурна варіантів використання.

7) Схема структурна концептуальної моделі предметного середовища

8) Схема взаємодії об’єктів.

9) Схема структурна компонент.

10) Схема структурна класів програмного забезпечення.

11) Схема структурна станів інтерфейсу.

12) Креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк,	Звітність
1.	Вивчення літератури за тематикою проекту	17.03.2020	Підготовка аналізу аналогів
2.	Розробка технічного завдання	24.03.2020	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	27.03.2020	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	03.04.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5.	Програмна реалізація програмного забезпечення	10.04.2020	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	17.04.2020	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	24.04.2020	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проекту	30.04.2020	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	08.05.2020	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ПРОБЛЕМНО-ОРІЄНТОВНА МОВА ДЛЯ ПРОГРАМУВАННЯ ЗАДАЧ З БІОІНФОРМАТИКИ

Програма та методика тестування

КПІ.ІП-6111.045490.03.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ І.В. Баклан

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____В.М. Каджая

Київ – 2020 року

ЗМІСТ

1 ОБ'ЄКТ ВИПРОБУВАНЬ

Мова програмування Kujira створений на мові Python та C++ з використанням бібліотеки lark.

2 МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

– функціональна працездатність вбудованих функцій мови Kujira;

– перевірка синтаксичних помилок;

– відповідність результатів після проведення арифметичних операцій або після виклику функцій;

– коректне імпортування модулів;

– перевірка працездатності функцій для біоінформатичних задач

– відповідність розробленого інтерпретатора описаним синтаксису та семантиці мови програмування;

– відповідність вимогам Технічного завдання.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як вхідна програма (код), так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

– функціональне тестування, зокрема на рівні Critical path test (ба-зове тестування);

– модульне тестування (Unit test) окремих модулів;

– системне тестування (System test);

– тестування синтаксису (Syntax testing);

– тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності).

Також було проведено базове інтеграційне тестування на сумісну роботу інтерпреатору.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами інструментарію assert.

Працездатність мови перевіряється шляхом:

– динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;

– динамічного ручного тестування на відповідність функціональним вимогам;

– статичного тестування коду.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ПРЕДМЕТНО-ОРІЄНТОВНА МОВА ДЛЯ ПРОГРАМУВАННЯ ЗАДАЧ З БІОІНФОРМАТИКИ

Керівництво програміста

КПІ.ІП-6111.045490.04.33

“ПОГОДЖЕНО”

Керівник проєкту:

_____ І.В. Баклан

Нормоконтроль:

_____ К.І. Ліщук.

Виконавець:

_____В.М. Каджая.

Київ – 2020 року

ЗМІСТ

1 ЗАГАЛЬНИЙ ОПИС МОВИ ПРОГРАМУВАННЯ KUJIRA

Kujira – це гибрідна мова програмування для програмування задач з біоінформатики, а також частково для програмування загального призначення. За синтаксисом схожий на мову Python та частково на C++.

1.1 Система типів

Kujira має динамічну типізацію. Мова має основні типи даних, а також основні арифметичні опреації над примітивними типами. Приклади наведені нижче:

```
text = “Hello, world!”
number = 12
//adding two numbers
sum = 12 + 24
//concatenation
res_con = concat(“Hello”, “world”)
```

Kujira також має структуру даних. Як і Python Kujira не має масивів, але вона замінена списком. Приклади структур даних наведені нижче:

```
//list(array)
li = [1, 2, 3, 4]
// set
s = {1, 2, 3, 4}
//dictionary(map)
dict.dictint(2).set(“num1”:1).set(“num2”:2) get()
```

1.2 Умовний оператор

```
sum = 2+2
a = 5
if sum equals a then
    print “true”
else
```

```
    print "false"
```

1.3 Цикли

```
for i in 1 .. 5:
    print i
```

2 ПРИКЛАД ПРОСТОЇ ПРОГРАМИ НА МОВІ KUJIRA

У даному розділі наведено приклади роботи з вбудованими функціями для аналізу у біоінформатиці та прості приклади роботи з мовою Kujira.

```
//function
sum = 2+2
void ex() {
    res = 5 + sum;
    print “Example”
}
ex()
// count dna
count_dna(“ATGCGGACCTAT”, “C”)
//reverse complement
reverse_complement(“TCGGinsGCCC”)
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ПРОБЛЕМНО-ОРІЄНТОВНА МОВА ДЛЯ ПРОГРАМУВАННЯ ЗАДАЧ З БІОІНФОРМАТИКИ

Опис програми

КПІ.ІП-6111.045490.03.13

“ПОГОДЖЕНО”

Керівник проєкту:

_____ І.В. Баклан

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____В.М. Каджая

Київ – 2020 року

Тексти програмного коду

Проблемно-орієнтовна мова для програмування задач з біоінформатики

(Найменування програми (документа))

DVD-R

(Вид носія даних)

14 арк, 40 Мб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2020

kujira.py

```
from lark import Lark, Transformer, Visitor, Tree, v_args, UnexpectedInput, Token
from lib.proglang.List import List
from lib.proglang.Dictionary import Dictionary
from lib.proglang.Set import Set
from lib.mathlib import mathlib
from lib.sortinglib import sortinglib
from lib.bioinfo import bioinfo
from lib.filereader import filereader
from lib.filewriter import filewriter
from lib.proglang.bigint import bigint
from lib.proglang.rational.Fraction import Fraction
import sys
import importlib
import importfile
import re
import ast
grammar = '''
?start: main

?main: calc | NAME "=" calc -> assign | value | printval | functype | funcnontype | pow_ | ceil_ | acos_ | asin_ | atan_ | cos_ | exp_
| fabs_ | floor_ | sin_ | tan_ | sqrt_ | log_ | log10_ | importingfile | NEWLINE | condition | diction | writeio | bignumsum |
bignumsub | bignummul | bignumpow | bioinfo | forloop | returning

?codetype: calc | NAME "=" calc -> assign | value | printval | functype | funcnontype | pow_ | ceil_ | acos_ | asin_ | atan_ | cos_ |
exp_ | fabs_ | floor_ | sin_ | tan_ | sqrt_ | log_ | log10_ | importingfile | NEWLINE | condition | diction | writeio | bignumsum |
bignumsub | bignummul | bignumpow | bioinfo | forloop | returning

?calc: prod | calc "+" prod -> add | calc "-" prod -> sub
?prod: atom | prod "*" atom -> mul | prod "/" atom -> div

?atom: NUMBER -> number | "-" atom -> neg | NAME -> var | "("calc")"

?value: array | SIGNED_NUMBER -> number | string | sorting | sets

?printval: "print" codetype | "print" NAME_VAL

?functype: "func" NAME "(" ")" "{" [codetype (";" codetype)*] ";" returning codetype "}" -> functypeassign | functypeatom
?type: "Int" | "String" | "Float" | "Decimal"
?functypeatom: "typed" NAME "(" ")" -> functypevar
?returning: "return"

?funcnontype: "void" NAME "(" ")" "{" [codetype (";" codetype)*] "}" -> funcassign | funcatom
?funcatom: NAME "(" ")" -> funcvar

?array: "[" [value ("," value)*] "]"

?sets: "{" [value ("," value)*] "}"

?diction: "dict" "." "dictint" "(" value ")" "." ["set" "(" value "," value ")"*] "." "get" "(" ")"

?writeio: "writeio" "." "open" "(" value "," value ")"

?readio: "readio" "." "open" "(" value ")"

?bignumsum: "bigint" "." "sum" "(" value "," value ")"
?bignumsub: "bigint" "." "sub" "(" value "," value ")"
?bignummul: "bigint" "." "mul" "(" value "," value ")"
?bignumpow: "bigint" "." "pow" "(" value "," value ")"

?bioinfo: count_dna | generate_dna_sequence | dna_frequency_map | read_dnafile | translate_to_protein | mutate |
reverse_complement | count_non_dna_bases_seq | dna_concat | sequence_alignment | motif | protein_mass | dna_to_rna |
count_mut
?count_dna: "count_dna" "(" value "," value ")"
?generate_dna_sequence: "generate_dna_sequence" "(" value ")"
?dna_frequency_map: "dna_frequency_map" "(" value "," value ")"
```

```
?read_dnafile: "read_dnafile" "(" value ")"
?translate_to_protein: "translate_to_protein" "(" value ")"
?mutate: "mutate" "(" value "," value "," value ")"
?reverse_complement: "reverse_complement" "(" value ")"
?count_non_dna_bases_seq: "count_non_dna_bases_seq" "(" value "," value ")"
?dna_concat: "dna_concat" "(" value "," value ")"
?sequence_alignment: "sequence_alignment" "(" value "," value "," value ")"
?motif: "motif" "(" value "," value ")"
?protein_mass: "protein_mass" "(" value ")"
?dna_to_rna: "dna_to_rna" "(" value ")"
?count_mut: "count_mut" "(" value "," value ")"

condition: "if" [statement "then" result ("elseif" statement ":" result)*] | "if" [statement "then" result ("elseif" statement ":"
result)*] "else" result
if: "if"
then: "then"
elseif: "elseif"
else: "else"
statement: expression
result: printval | condition
expression: (VARIABLE | SIGNED_NUMBER) action_operator (VARIABLE | SIGNED_NUMBER)
VARIABLE: /[a-zA-Za-яА-Я0-9_.-]+/
?action_operator: ACTION_OPERATOR
ACTION_OPERATOR: "<"|">"|"="|"is"|">="|"<="|"!="|"in"|"equals"

?forloop: "for" NAME "in" value ".." value ":"  [codetype (";" codetype)*]

?pow_ : "pow" "(" NUMBER "," NUMBER ")"
?ceil_ : "ceil" "(" NUMBER ")"
?acos_ : "acos" "(" NUMBER ")"
?asin_ : "asin" "(" NUMBER ")"
?atan_ : "atan" "(" NUMBER ")"
?cos_ : "cos" "(" NUMBER ")"
?exp_ : "exp" "(" NUMBER ")"
?fabs_ : "fabs" "(" NUMBER ")"
?floor_ : "floor" "(" NUMBER ")"
?sin_ : "sin" "(" NUMBER ")"
?tan_ : "tan" "(" NUMBER ")"
?sqrt_ : "sqrt" "(" NUMBER ")"
?log_ : "log" "(" NUMBER ")"
?log10_ : "log10" "(" NUMBER ")"

?sorting : "sort" "(" "[" [value ("," value)*] "]" ")"

importingfile: "import" NAME NAME  NUMBER "," NUMBER

string : ESCAPED_STRING

NAME_VAL: NAME

COMMENT: "//" /(.|\\n|\\r)*/

%import common.NUMBER
%import common.SIGNED_NUMBER
%import common.ESCAPED_STRING
%import common.CNAME -> NAME
%import common.WS_INLINE
%import common.NEWLINE
%ignore " "
%ignore WS_INLINE
%ignore COMMENT
'''

@v_args(inline=True)
class LanguageTransformer(Transformer):
   from operator import add, sub, mul, truediv as div, neg
   number = float
```

```
def __init__(self):
    self.vars = {}
    self.funcvars = {}
    self.functypevars = {}
def assign(self, name, args):
    self.vars[name] = args
    return args
def functypeassign(self, name, *args):
    for i in args:
        self.functypevars[name] = args
    return args
def functypevar(self, name):
    return self.functypevars[name]
def funcassign(self, name, *args):
    for i in args:
        self.funcvars[name] = args
    return args
def funcvar(self, name):
    return self.funcvars[name]
def var(self, name):
    return self.vars[name]
def array(self, *elements):
    l = List.List()
    l.createnode(int(elements[0]))
    for arg in elements:
        l.insert_start(int(arg))
    l.display()
def sets(self, *elements):
    s = Set.Set()
    for arg in elements:
        s.put(int(arg))
    s.display()
def pow_(self, elem, elem2):
    return mathlib.pow_(int(elem), int(elem2))
def ceil_(self, elem):
    return mathlib.ceil_(int(elem))
def acos_(self, elem):
    return mathlib.acos_(int(elem))
def asin_(self, elem):
    return mathlib.asin_(int(elem))
def atan_(self, elem):
    return mathlib.atan_(int(elem))
def cos_(self, elem):
    return mathlib.cos_(int(elem))
def exp_(self, elem):
    return mathlib.exp_(int(elem))
def fabs_(self, elem):
    return mathlib.fabs_(int(elem))
def floor_(self, elem):
    return mathlib.floor_(int(elem))
def sin_(self, elem):
    return mathlib.sin_(int(elem))
def tan_(self, elem):
    return mathlib.tan_(int(elem))
def sqrt_(self, elem):
    return mathlib.sqrt_(int(elem))
def log_(self, elem):
    return mathlib.log_(int(elem))
def log10_(self, elem):
    return mathlib.log_10(int(elem))
def sorting(self, *elem):
    a = []
    for arg in elem:
        a.append(int(arg))
    a = tuple(a)
    return sortinglib.arr(a)
def importingfile(self, arg, arg2, arg3, arg4):
```

```
    importfile.imports(str(arg), str(arg2), str(arg3), str(arg4))
  def condition(self, *expr):
    num1 = None
    num2 = None
    true_ = ""
    false_ = ""
    sign = ""
    for expp in expr[1].children:
      true_ = expp.children[0].value
    for expp in expr[2].children:
      false_ = expp.children[0].value
    for expr_val in expr[0].children:
      if expr_val.children[0].type == 'VARIABLE':
        num1 = int(self.var(str(expr_val.children[0].value)))
      else:
        num1 = int(expr_val.children[0].value)
      sign = expr_val.children[1].value
      if expr_val.children[2].type == 'VARIABLE':
        num2 = int(self.var(str(expr_val.children[2].value)))
      else:
        num2 = int(expr_val.children[2].value)
    if sign == '<':
      if num1 < num2:
        return true_
      else:
        return false_
    elif sign == '>':
      if num1 > num2:
        return true_
      else:
        return false_
    elif sign == 'is':
      if num1 is num2:
        return true_
      else:
        return false_
    elif sign == 'equals':
      if num1 == num2:
        return true_
      else:
        return false_
    elif sign == '!=':
      if num1 != num2:
        return true_
      else:
        return false_
    elif sign == '>=':
      if num1 >= num2:
        return true_
      else:
        return false_
    elif sign == '<=':
      if num1 <= num2:
        return true_
      else:
        return false_
    else:
      return false_
  def diction(self, *elements):
    d = Dictionary.dictint(int(elements[0]))
    for arg in elements:
      d.set(int(arg), int(arg))
    d.get()
  def writeio(self, *ele,elements):
    filewriter.write_file(elements[0], elements[1])
  def readio(self, elem):
    print("->",elem)
```

```
        a = None
        for a in elem:
            a = elem
        print("bn", a)
        filereader.read_file(a)
    def bignumsum(self, *elem):
        num1 = None
        num2 = None
        for i in elem[0].children:
            num1 = i.value
        for i in elem[1].children:
            num2 = i.value
        num1 = num1.replace("", '')
        num2 = num2.replace("", '')
        print("->", bigint.BigInt().addition(num1, num2))
    def bignumsub(self, *elem):
        num1 = None
        num2 = None
        for i in elem[0].children:
            num1 = i.value
        for i in elem[1].children:
            num2 = i.value
        num1 = num1.replace("", '')
        num2 = num2.replace("", '')
        print("-_->", bigint.BigInt().substraction(num1, num2))
    def bignummul(self, *elem):
        num1 = None
        num2 = None
        for i in elem[0].children:
            num1 = i.value
        for i in elem[1].children:
            num2 = i.value
        num1 = num1.replace("", '')
        num2 = num2.replace("", '')
        print('-->', bigint.BigInt().multiply(num1, num2))
    def bignumpow(self, *elem):
        num1 = None
        num2 = None
        for i in elem[0].children:
            num1 = i.value
        for i in elem[1].children:
            num2 = i.value
        num1 = num1.replace("", '')
        num2 = num2.replace("", '')
        bigint.BigInt().pow(int(num1), int(num2))
    def count_dna(self, *elem):
        dna = None
        base = None
        for i in elem[0].children:
            dna = i.value
        for i in elem[1].children:
            base = i.value
        dna = dna.replace("", '')
        base = base.replace("", '')
        bioinfo.count_dna(dna, base)
    def generate_dna_sequence(self, *elem):
        length = None
        for i in elem[0].children:
            length = i.value
        length = length.replace("", '')
        bioinfo.generate_dna_sequence(int(length))
    def dna_frequency_map(self, *elem):
        dna = None
        k = None
        for i in elem[0].children:
            dna = i.value
        for i in elem[1].children:
```

```
            k = i.value
        dna = dna.replace("", '')
        k = k.replace("", '')
        bioinfo.dna_frequency_map(dna, int(k))
    def read_dnafile(self, *elem):
        filename = None
        for i in elem[0].children:
            filename = i.value
        filename = filename.replace("", '')
        bioinfo.read_dnafile(filename)
    def translate_to_protein(self, *elem):
        filename = None
        for i in elem[0].children:
            filename = i.value
        filename = filename.replace("", '')
        bioinfo.translate_to_protein(filename)
    def mutate(self, *elem):
        dna = None
        mutation = None
        threshold = None
        for i in elem[0].children:
            dna = i.value
        for i in elem[1].children:
            mutation = i.value
        for i in elem[2].children:
            threshold = i.value
        dna = dna.replace("", '')
        mutation = mutation.replace("", '')
        threshold = threshold.replace("", '')
        mutation = ast.literal_eval(mutation)
        bioinfo.mutate(dna, mutation, threshold)
    def reverse_complement(self, *elem):
        seq = None
        for i in elem[0].children:
            seq = i.value
        seq = seq.replace("", '')
        bioinfo.reverse_complement(seq)
    def count_non_dna_bases_seq(self, *elem):
        seq = None
        allowed_bases = None
        for i in elem[0].children:
            seq = i.value
        for i in elem[1].children:
            allowed_bases = i.value
        seq = seq.replace("", '')
        allowed_bases = list(allowed_bases.replace("", ''))
        bioinfo.count_non_dna_bases_seq(seq, allowed_bases=allowed_bases)
    def dna_concat(self, *elem):
        dna1 = None
        dna2 = None
        for i in elem[0].children:
            dna1 = i.value
        for i in elem[1].children:
            dna2 = i.value
        dna1 = dna1.replace("", '')
        dna2 = dna2.replace("", '')
        bioinfo.dna_concat(dna1, dna2)
    def sequence_alignment(self, *elem):
        pass
    def motif(self, *elem):
        s = None
        t = None
        for i in elem[0].children:
            s = i.value
        for i in elem[1].children:
            t = i.value
        s = s.replace("", '')
```

```
        t = t.replace('"', '')
        bioinfo.motif(s, t)
    def protein_mass(self, *elem):
        pass
    def dna_to_rna(self, *elem):
        dna = None
        for i in elem[0].children:
            dna = i.value
        dna = dna.replace('"', '')
        bioinfo.dna_to_rna(dna)
    def count_mut(self, *elem):
        s = None
        t = None
        for i in elem[0].children:
            s = i.value
        for i in elem[1].children:
            t = i.value
        s = s.replace('"', '')
        t = t.replace('"', '')
        bioinfo.count_mut(s, t)
    def forloop(self, *elem):
        obj1 = int(elem[1])
        obj2 = int(elem[2])
        obj3 = None
        lis = []
        #id_obj = {}
        #print(elem)
        if elem[0].type == 'NAME' and type(elem[3]) == Tree:
            for i in elem[3].children:
                obj3 = i.value
            obj3 = obj3.replace('"', '')
            for i in range(obj1, obj2):
                lis.append(obj3)
        else:
            #obj3 = int(elem[3])
            # print(elem[3])
            # print(elem[3].data)
            # print(elem[3].children)
            # for i in elem[3].children:
            #     print(i)
            #     obj3 = i.value
            # obj3 = obj3.replace('"', '')
            for i in range(obj1, obj2):
                lis.append(i)
        return lis

l = Lark(grammar, parser='lalr', transformer=LanguageTransformer())
#print(l.parse('''sum = 2+2'''))
#print(l.parse('''a = 5'''))
#print(l.parse('''if sum equals a then print "true" else print "false"'''))

#print(l.parse('''a = 5'''))
#print(l.parse('''if a > 12 then print "true" else print "false"'''))

#print(l.parse('''if 12 equals 12 then print "true" else print "false"'''))

#print(l.parse('''bigint.sum("2", "2")'''))

#print(l.parse('''sum = 2+2'''))
#print(l.parse('''void ex() { print "Hello"; c=2+2 }'''))
#print(l.parse('''ex()'''))

#print(l.parse('''sum = 2+2'''))
#print(l.parse('''print sum'''))

#print(l.parse('''i = 0'''))
#print(l.parse('''for i in 1 .. 5: print i'''))
```

```
#print(l.parse('''func ex() { print "Hello"; c=2+2; return c }'''))
#print(l.parse('''typed ex()'''))

#print(l.parse('''i = 0'''))
#print(l.parse('''void ex() { for i in 1 .. 5: print i }'''))
#print(l.parse('''ex()'''))


class KujiraSyntaxError(SyntaxError):
   def __init__(self, label, line, column):
      self.label = label
      self.line = line
      self.column = column
   def __str__(self):
      #context, line, column = self.args
      return '%s at line %s, column %s.' % (self.label, self.line, self.column)

class KujiraMissingValue(KujiraSyntaxError):
   label = 'Missing Value'

class KujiraMissingOpening(KujiraSyntaxError):
   label = 'Missing Opening'

class KujiraMissingClosing(KujiraSyntaxError):
   label = 'Missing Closing'

class KujiraMissingComma(KujiraSyntaxError):
   label = 'Missing Comma'

class KujiraMissingScope(KujiraSyntaxError):
   label = 'Missing Scope'

class KujiraMissingPaws(KujiraSyntaxError):
   label = 'Missing Paws'

class KujiraWrongCalculation(KujiraSyntaxError):
   label = 'Wrong Calculation'

class KujiraObjNotDefined(KujiraSyntaxError):
   label = 'Object Not Defined'

class KujiraInnerFunctonError(KujiraSyntaxError):
   label = 'Inner Functon Error'

class KujiraIfThenElseError(KujiraSyntaxError):
   label = 'If-Then-Else Error'

class KujiraListTypeDiffError(KujiraSyntaxError):
   label = 'List Different Type Error'

class KujiraImportSyntaxError(KujiraSyntaxError):
   label = 'Kujira Import Syntax Error'

def parss(text):
   try:
      j = l.parse(text)
      print(j)
   except UnexpectedInput as i:
      xc_class = i.match_examples(l.parse, {
         KujiraMissingClosing : ['[1,2,3,4', '[1,2,3,4,'],
         KujiraMissingOpening : ['1,2,3,4]', '1,2,3,4,]'],
         KujiraMissingComma   : ['[1 2 3 4]'],
         KujiraMissingValue   : ['a = ', 'a= '],
         KujiraMissingScope   : ['func (', 'sort(', 'func )'],
         KujiraMissingPaws    : ["print 'Hello'"],
         KujiraWrongCalculation : ["'2' + 2", "cd = '2' + 34"],
```

```
        KujiraObjNotDefined  : ['acd'],
        KujiraInnerFunctonError : ['void a() { void printH() { print "Hello" } } printH()'],
        KujiraIfThenElseError: ['iff a < 12 then print "true" else print "false"'],
        KujiraListTypeDiffError: ['[1, 2, 3, "a"]'],
        KujiraImportSyntaxError: ['importttt mathlib cos_ 12'],
      })
      if not xc_class:
        raise
      raise xc_class(i.get_context(text), i.line, i.column)

def test():
  try:
    parss('''importttt mathlib cos_ 12''')
    #parss('''print "Hello world"''')
  except KujiraMissingClosing as e:
    print(e)
  except KujiraMissingOpening as e:
    print(e)
  except KujiraMissingComma as e:
    print(e)
  except KujiraMissingValue as e:
    print(e)
  except KujiraMissingScope as e:
    print(e)
  except KujiraMissingPaws as e:
    print(e)
  except KujiraWrongCalculation as e:
    print(e)
  except KujiraInnerFunctonError as e:
    print(e)
  except KujiraObjNotDefined as e:
    print(e)
  except KujiraIfThenElseError as e:
    print(e)
  except KujiraListTypeDiffError as e:
    print(e)
  except KujiraImportSyntaxError as e:
    print(e)

test()
```

bioinfo.py

```
import random

# dna count
def count_dna(dna, base):
  print('dna', dna)
  print('base', base)
  i = 0 # counter
  for c in dna:
    print('c:', c)
    if c == base:
      print('True if test')
      i += 1
  return i

# generating dna random sequence
#length -> int
def generate_dna_sequence(length):
  return ''.join(random.choice(list('ACTG')) for _ in range(length))

# making dna frequency
# dna -> str
#k -> int
def dna_frequency_map(dna, k):
  freq = {}
```

```
    n = len(dna)
    for i in range(n-k+1):
        pattern = dna[i:i+k]
        if pattern not in freq:
            freq[pattern] = 1
        else:
            freq[pattern] += 1
    return freq

# reading dna from file
#filename -> str
def read_dnafile(filename):
    lines = open(filename, 'r').readlines()
    # Remove newlines in each line and join
    dna = ''.join([line.strip() for line in lines])
    return dna

# dna to protein
#filename -> str
def translate_to_protein(filename):
    f = open(filename, 'r')
    seq = f.read()
    seq = seq.replace("\n", "")
    seq = seq.replace("\r", "")

    table = {
        'ATA': 'I', 'ATC': 'I', 'ATT': 'I', 'ATG': 'M',
        'ACA': 'T', 'ACC': 'T', 'ACG': 'T', 'ACT': 'T',
        'AAC': 'N', 'AAT': 'N', 'AAA': 'K', 'AAG': 'K',
        'AGC': 'S', 'AGT': 'S', 'AGA': 'R', 'AGG': 'R',
        'CTA': 'L', 'CTC': 'L', 'CTG': 'L', 'CTT': 'L',
        'CCA': 'P', 'CCC': 'P', 'CCG': 'P', 'CCT': 'P',
        'CAC': 'H', 'CAT': 'H', 'CAA': 'Q', 'CAG': 'Q',
        'CGA': 'R', 'CGC': 'R', 'CGG': 'R', 'CGT': 'R',
        'GTA': 'V', 'GTC': 'V', 'GTG': 'V', 'GTT': 'V',
        'GCA': 'A', 'GCC': 'A', 'GCG': 'A', 'GCT': 'A',
        'GAC': 'D', 'GAT': 'D', 'GAA': 'E', 'GAG': 'E',
        'GGA': 'G', 'GGC': 'G', 'GGG': 'G', 'GGT': 'G',
        'TCA': 'S', 'TCC': 'S', 'TCG': 'S', 'TCT': 'S',
        'TTC': 'F', 'TTT': 'F', 'TTA': 'L', 'TTG': 'L',
        'TAC': 'Y', 'TAT': 'Y', 'TAA': '_', 'TAG': '_',
        'TGC': 'C', 'TGT': 'C', 'TGA': '_', 'TGG': 'W',
    }
    protein = ''
    if len(seq)%3 == 0:
        for i in range(0, len(seq)):
            codon = seq[i: i+3]
            protein += table[codon]
    return protein

# generate dna mutation
#dna -> str
#mutation -> dict exmp ( {"A" : "T"})
#threshold -> int
def mutate(dna, mutation, threshold):
    dna = list(dna)
    for index, char in enumerate(dna):
        if char in mutation:
            if random.random() < threshold:
                dna[index] = mutation[char]
    return ''.join(dna)

# dna reverse complement
'''
    **How to use**
    seq = "TCGGinsGCCC"
    print ("Reverse Complement:")
```

```
    print(reverse_complement(seq))
'''
#seq -> str
alt_map = {'ins':'0'}
complement = {'A': 'T', 'C': 'G', 'G': 'C', 'T': 'A'}

def reverse_complement(seq):
    for k,v in alt_map.items():
        seq = seq.replace(k,v)
    bases = list(seq)
    bases = reversed([complement.get(base,base) for base in bases])
    bases = ''.join(bases)
    for k,v in alt_map.items():
        bases = bases.replace(v,k)
    return bases

# count non-DNA bases in a sequence
'''
** How to use**
print(count_dna("ACTRGATCYGATCGANTCGATG"))
print(count_dna("ACTRGATCYGATCGANTCGATG", ['A','T','C','G','N']))
print(count_dna("actgratcygtgancttttgacg"))
'''
#seq -> str
#allowed_bases -> list
def count_non_dna_bases_seq(seq, allowed_bases=['A','T','G','C']):
    seq = seq.upper()
    total_dna_bases = 0
    for base in allowed_bases:
        total_dna_bases = total_dna_bases + seq.count(base.upper())
    dna_fraction = total_dna_bases / len(seq)
    return(dna_fraction * 100)

# dna concatenate
# dna1 -> str
# dna2 -> str
def dna_concat(dna1, dna2):
    return dna1 + dna2

# dna sequence alignment
# score ->
# seq1 ->
# seq2 ->
def sequence_alignment(score, seq1, seq2):
    alphabet = ['A', 'C', 'G', 'T']
    D = []
    for i in range(len(seq1)+1):
        D.append([0] * (len(seq2)+1))
    for i in range(1, len(seq1)+1):
        D[i][0] = D[i-1][0] + score[alphabet.index(seq1[i-1])][i-1]
    for i in range(1, len(seq2)+1):
        D[0][i] = D[0][i-1] + score[-1][alphabet.index(seq2[i-1])]
    for i in range(1, len(seq1) + 1):
        for j in range(1, len(seq2) + 1):
            distHor = D[i][j-1] + score[-1][alphabet.index(seq2[j-1])]
            distVer = D[i][j] + score[alphabet.index(seq1[i-1])][-1]
            if seq1[i-1] == seq2[j-1]:
                distDiag = D[i-1][j-1]
            else:
                distDiag = D[i-1][j-1] + score[alphabet.index(seq1[i-1])][alphabet.index(seq2[j-1])]
            D[i][j] = min(distHor, distVer, distDiag)
    return D[-1][-1]

# find motif in dna sequence
# s -> str
# t -> str
def motif(s, t):
```

```
    result = []
    len1 = len(s)
    len2 = len(t)
    for i in range(0, len1 - len2 + 1):
        if s[i:i+len2] == t:
            result.append(i+1)
    return result

# protein mass calculator
def protein_mass(pmass):
    mass_table = {
        'A':  71.03711,  'C':  103.00919,  'D':  115.02694,
        'E':  129.04259, 'F':  147.06841,  'G':  57.02146,
        'H':  137.05891, 'I':  113.08406,  'K':  128.09496,
        'L':  113.08406, 'M':  131.04049,  'N':  114.04293,
        'P':  97.05276,  'Q':  128.05858,  'R':  156.10111,
        'S':  87.03203,  'T':  101.04768,  'V':  99.06841,
        'W':  186.07931, 'Y':  163.06333,
    }
    n = 0
    for i in pmass:
        n += mass_table[i]
    return n

# dna to rna
# dna -> str
def dna_to_rna(dna):
    return dna.replace('T', 'U')

# count point mutation
# s -> str
# t -> str
def count_mut(s, t):
    dh = 0
    for i, c in enumerate(s):
        if c != t[i]:
            dh += 1
    return dh
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ПРОБЛЕМНО-ОРІЄНТОВНА МОВА ДЛЯ ПРОГРАМУВАННЯ ЗАДАЧ З БІОІНФОРМАТИКИ

Графічні матеріали

КПІ.ІП-6111.045490.06.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ І.В. Баклан

Нормоконтроль:

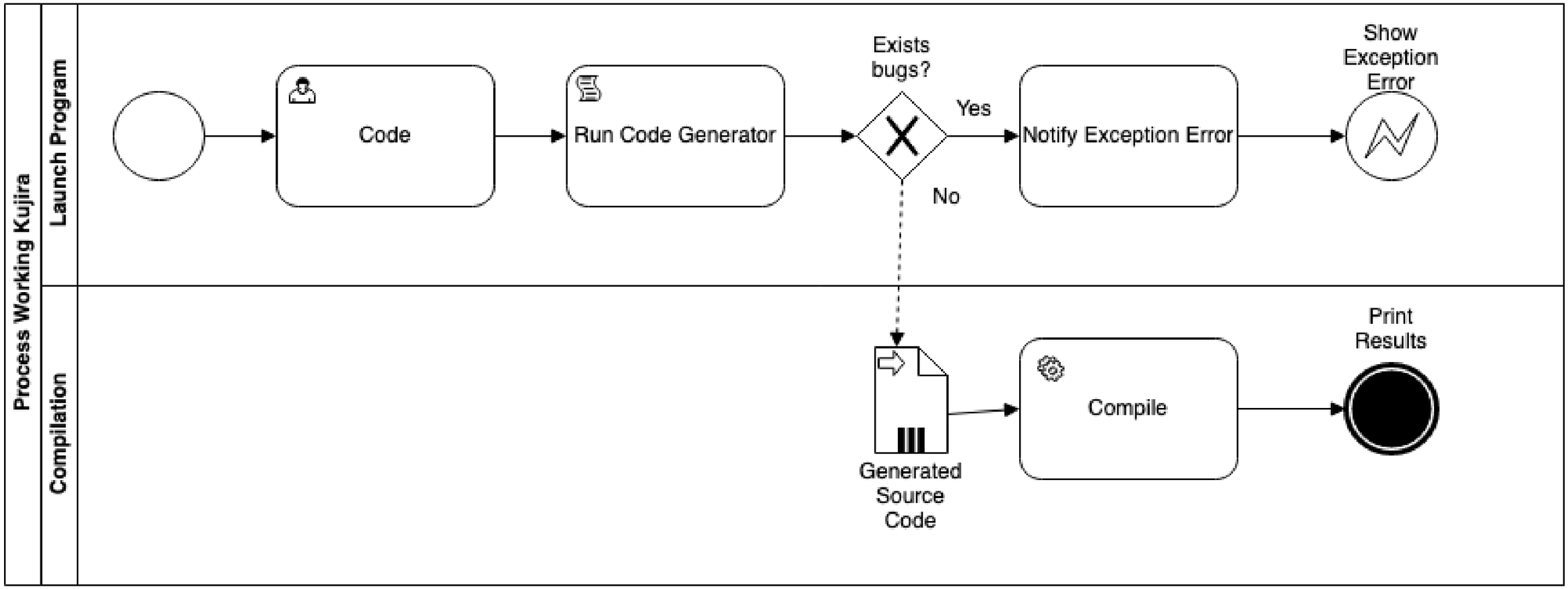
_____ К.І. Ліщук

Виконавець:

_____ В.М.Каджая

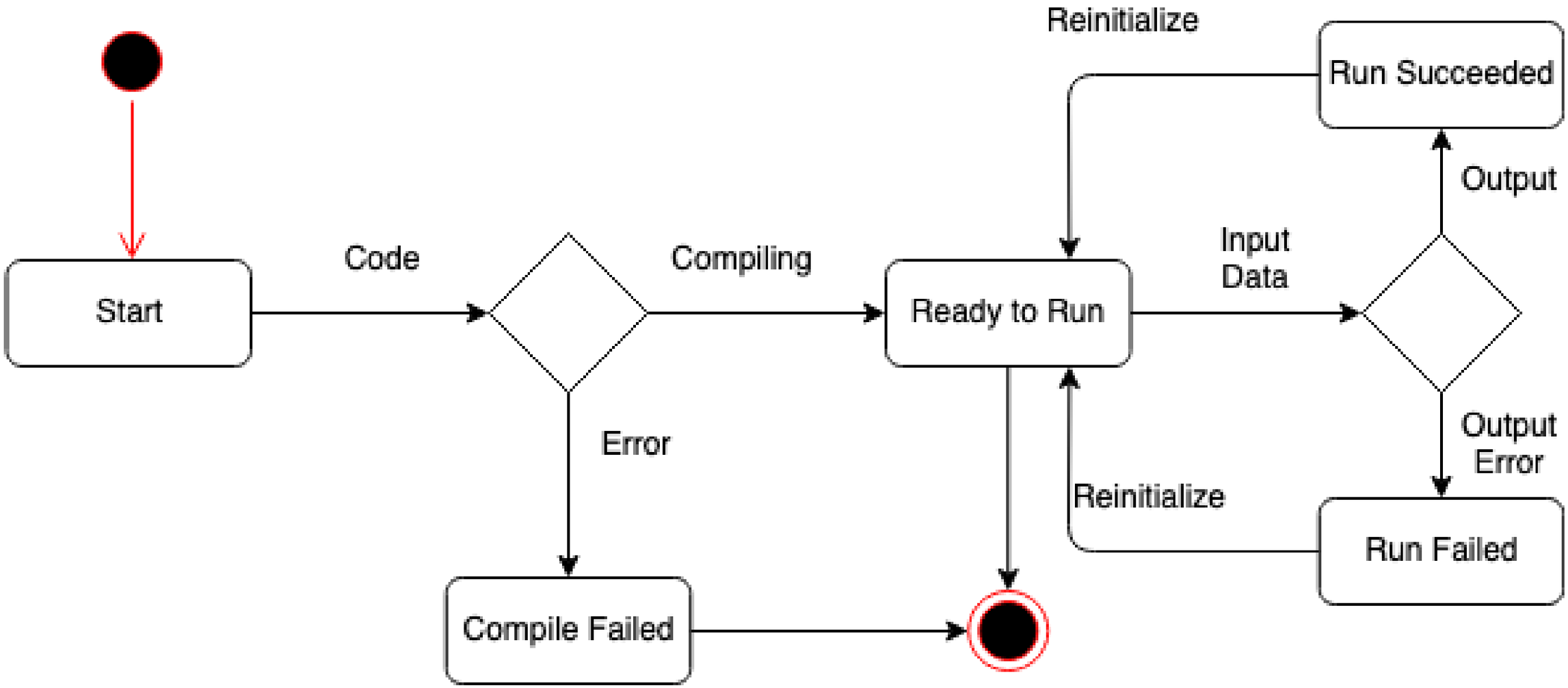
Київ – 2020 року

КПІ.ІП-6111.045490.06.99.СБП

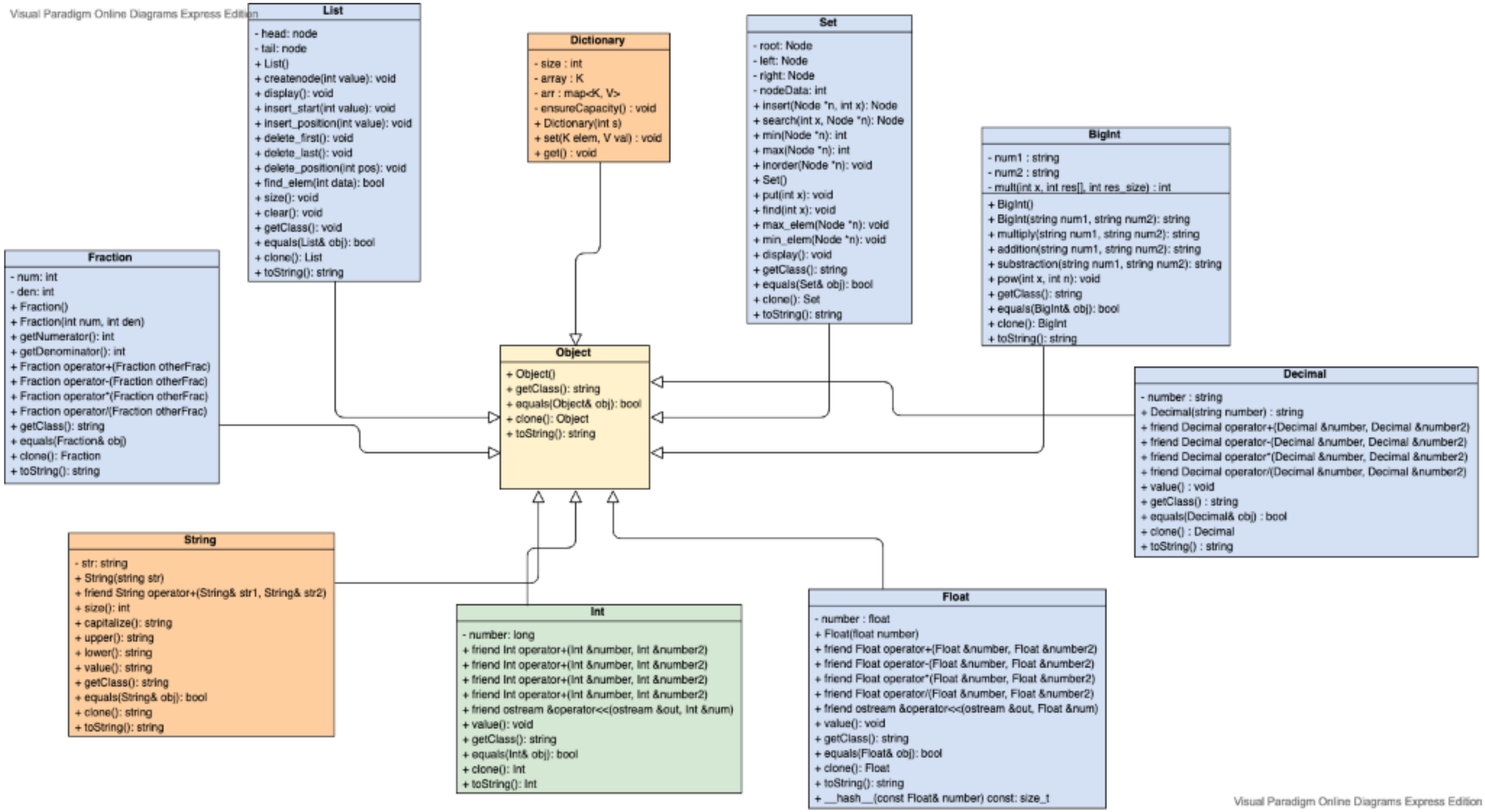


					КПІ.ІП-6111.045490.06.99.СБП		
						Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Схема бізнес-процесу			
Розробив		Каджая В.М.				Аркуш	Аркушів	
Перевірив		Баклан І.В.			Проблемно-орієнтовна мова для програмування задач з біоінформатіки	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-61		
Т. кон.								
Н. кон.		Ліщук К.І.						
Затвердив								

КПІ.ІП-6111.045490.06.99.СС



					КПІ.ІП-6111.045490.06.99.СС		
					Схема структурна станів системи	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Каджая В.М.					
Перевірив		Баклан І.В.				Аркуш	Аркушів
Т. кон.							
Н. кон.		Ліщук К.І.			Проблемно-орієнтовна мова для програмування задач з біоінформатики	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-61	
Затвердив							



					КПІ.ІП-6111.045490.06.99.СС	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів програмного забезпечення	Літера / Маса / Масштаб
Розробив		Каджая В.М.				
Перевірив		Баклан І.В.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Ліщук К.І.			Проблемно-орієнтовна мова для програмування задач з біоінформатики	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-61
Затвердив						